ON COMPUTATION AND APPLICATION OF OPTIMAL TRANSPORT

A Dissertation
Presented to
The Academic Faculty

By

Yujia Xie

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
Computational Science and Engineering
College of Computing

Georgia Institute of Technology

August 2021

ON COMPUTATION AND APPLICATION OF OPTIMAL TRANSPORT

Thesis committee:

Dr. Hongyuan Zha, Advisor
School of Computational Science and Engineering
Georgia Institute of Technology

Dr. Chao Zhang
School of Computational Science and Engineering
Georgia Institute of Technology

Dr. Tuo Zhao, Co-advisor
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Yongxin Chen
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Xiaojing Ye
Department of Mathematics and Statistics
Georgia State University

Date approved: July 2, 2021

The Heavens are in motion ceaselessly; The enlight'ned exert themselves constantly.

The Book of Change

To my spouse Jialei and my beloved parents.

ACKNOWLEDGMENTS

I had many fortunate moments in the past five years.

I would like to thank professor Vigor Yang for extending me the Ph.D. offer of Georgia Tech, for which I met this lovely city. I would like to thank professor Le Song for supervising me my first machine learning project, which is literally the very start of my career as an ML researcher.

Professor Hongyuan Zha became my advisor in the summer of 2017. In the past four years, he has provided continuously supports for my study and research, and constantly encouraged me to realize my full potential. I remember him congratulating me for my first paper accepted, even before I noticed it. I remember him encouraging me to stay on the right road and rise to the challenge, in face of unfair judgement. I cannot even imagine how I can come this far without him.

I started working with professor Tuo Zhao in the winter of 2018. He has taught me how to write a paper, how to explain things clearly, and his understanding of the current academic field. I remember many late nights we working together, many meetings with extensive discussions and reaching agreements, and many moments where I started to see the worlds from a different perspectives. Ever started from I worked with professor Tuo, my papers keep getting accepted.

I would like to thank the friends and colleagues who made this work possible – Xiangfeng Wang, Ruijia Wang, Minshuo Chen, Haoming Jiang, Feng Liu, Yixiu Mao, Simiao Zuo, Hongteng Xu, Jiachen Yang, Chen Liang, Tianyi Liu, Zhehui Chen, Yan Li, etc. Special thanks are due to Rakshit Trivedi, who warmed me up in the dark times and shared me many gold advice.

Additionally, I would like to thank the other members of my Ph.D. dissertation committee: professors Yongxin Chen, Chao Zhang, and Xiaojing Ye., for their time to attend my thesis defense and their insightful suggestions on my dissertation, which are of great

help.

Last but not the least, I feel deeply indebted to the selfless love and endless support of my parents. My husband, Jialei Chen, has companied me for thousands of days and nights, witnessed my ups and downs, and constantly urged me to become a better self. You and our parents are always the most important part of my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The Optimal Transport (OT) problem naturally arises in various machine learning problems, where one needs to align data from multiple sources. For example, the training data and application scenarios oftentimes have a domain gap, e.g., the training data is annotated photos collected in the daytime, yet the application scenario is in dark hours. In this case, we need to align the two datasets, so that the annotation information can be shared across them. During my Ph.D. study, I propose scalable algorithms for efficient OT computation, and its novel applications in end-to-end learning. Specifically,

1. For OT computation, I consider both discrete cases and continuous cases. For the discrete cases, I develop an Inexact Proximal point method for exact Optimal Transport problem (IPOT) with the proximal operator approximately evaluated at each iteration using projections to the probability simplex. The algorithm (a) converges to exact Wasserstein distance with theoretical guarantee and robust regularization parameter selection, (b) alleviates numerical stability issue, (c) has similar computational complexity to Sinkhorn, and (d) avoids the shrinking problem when apply to generative models. Furthermore, a new algorithm is proposed based on IPOT to obtain sharper Wasserstein barycenter.

For continuous case, I propose an implicit generative learning-based framework called SPOT (Scalable Push-forward of Optimal Transport). Specifically, we approximate the optimal transport plan by a pushforward of a reference distribution, and cast the optimal transport problem into a minimax problem. We then can solve OT problems efficiently using primal dual stochastic gradient-type algorithms.

2. To explore the connections between OT and end-to-end learning, I developed a differentiable top-k operator, and a differentiable permutation step.

For the top-k operation, i.e., finding the k largest or smallest elements from a collection of scores, is an important model component used in information retrieval, machine learning, and data mining. However, if the top-k operation is implemented in an algorithmic

way, e.g., using bubble algorithm, the resulting model cannot be trained in an end-to-end way using prevalent gradient descent algorithms. This is because these implementations typically involve swapping indices, whose gradient cannot be computed. Moreover, the corresponding mapping from the input scores to the indicator vector of whether this element belongs to the top-k set is essentially discontinuous. To address the issue, we propose a smoothed approximation, namely the SOFT (Scalable Optimal transport-based diFferenTiable) top-k operator. Specifically, our SOFT top-k operator approximates the output of the top-k operation as the solution of an Entropic Optimal Transport (EOT) problem. The gradient of the SOFT operator can then be efficiently approximated based on the optimality conditions of EOT problem. We apply the proposed operator to the k-nearest neighbors and beam search algorithms, and demonstrate improved performance.

For the differentiable permutation step, I connect optimal transport to a variant of regression problem, where the correspondence between input and output data is not available. Such shuffled data is commonly observed in many real world problems. Taking flow cytometry as an example, the measuring instruments may not be able to maintain the correspondence between the samples and the measurements. Due to the combinatorial nature of the problem, most existing methods are only applicable when the sample size is small, and limited to linear regression models. To overcome such bottlenecks, we propose a new computational framework – ROBOT – for the shuffled regression problem, which is applicable to large data and complex nonlinear models. Specifically, we reformulate the regression without correspondence as a continuous optimization problem. Then by exploiting the interaction between the regression model and the data correspondence, we develop a hypergradient approach based on differentiable programming techniques. Such a hypergradient approach essentially views the data correspondence as an operator of the regression, and therefore allows us to find a better descent direction for the model parameter by differentiating through the data correspondence. ROBOT can be further extended to the inexact correspondence setting, where there may not be an exact alignment between the

input and output data. Thorough numerical experiments show that ROBOT achieves better performance than existing methods in both linear and nonlinear regression tasks, including real-world applications such as flow cytometry and multi-object tracking.

CHAPTER 1

INTRODUCTION

The Optimal Transport (OT) problem naturally arises in a variety of machine learning applications, where we need to align data from multiple sources. One example is domain adaptation: We want to learn a model from a source dataset, e.g., annotated photos taken in day times, which can be further adapted to target datasets, e.g., unannotated photos taken in dark hours. In this case, we need to align the source dataset and the target datasets, so that the annotation can be shared across different datasets. Another example is resource allocation: We want to assign a set of assets (one data source) to a set of receivers (another data source) so that an optimal economic benefit is achieved. In this case, we need to find the optimal alignment between the assets and receivers.

The optimal transport problem has a long history, and its earliest literature dates back to [1]. Since then, it has attracted increasing attention and been widely studied in multiple communities such as applied mathematics, probability, economy, and geography. Specifically, we consider two sets of d-dimensional data, which are generated from two different distributions denoted by $X \sim \mu$ and $Y \sim \nu$. We aim to find an optimal joint distribution γ of X and Y, which minimizes the expectation on a cost function c, i.e.,

$$\gamma^* = \underset{\gamma \in \Pi(\mu,\nu)}{\operatorname{argmin}} \mathbb{E}_{(X,Y)\sim\gamma}[c(X,Y)], \tag{1.1}$$

The feasible set $\Pi(\mu,\nu)$ of γ requires the marginal distribution of X and Y in γ to be identical to μ and ν, respectively. Existing literature often refers to the optimal expected cost $\mathcal{W}^*(\mu,\nu) = \mathbb{E}_{(X,Y)\sim\gamma^*}[c(X,Y)]$ as *Wasserstein distance*, and γ^* as the *optimal transport plan*. For domain adaptation, the function c measures the discrepancy between X and Y, and the optimal transport plan γ^* essentially reveals the transfer of the knowledge

from source X to target Y. For resource allocation, the function c is the cost of assigning resource X to receiver Y, and the optimal transport plan γ^* yields the optimal assignment.

Despite the usefulness, the applications of OT have been largely hampered by its computational cost. During my Ph.D. study, I propose scalable algorithms for the efficient computation of OT in both discrete case and the continuous case, and also develop novel applications exploiting the nice geometric property of OT. Specifically, my Ph.D. research can be separated into the following three parts.

Computation – Discrete Case. In some applications, OT aligns two *discrete* distributions. Taking the resource allocation example, the storages of the assets and the capacities of the receivers can be viewed as parameters of two categorical distributions. In this case, OT amounts to solving the following optimization problem,

$$\mathbf{\Gamma}^* = \min_{\mathbf{\Gamma}} \langle \boldsymbol{C}, \mathbf{\Gamma} \rangle, \quad \text{subject to} \quad \mathbf{\Gamma} \mathbf{1}_n = \boldsymbol{\mu}, \quad \mathbf{\Gamma}^\top \mathbf{1}_n = \boldsymbol{\nu}. \tag{1.2}$$

Here $\boldsymbol{\mu}, \boldsymbol{\nu}$ are two probability vectors, matrix $\boldsymbol{C} = [c_{ij}] \in \mathbb{R}_+^{n \times n}$ is the *cost matrix*, whose element c_{ij} represents the distance between the i-th support point of $\boldsymbol{\mu}$ and the j-th one of $\boldsymbol{\nu}$. This is a linear programming problem with typical super $O(n^3)$ complexity [2].

In this part, my collaborators and I propose a new algorithm, Inexact Proximal point method for Optimal Transport (IPOT) to compute the optimal transport plan using generalized proximal point iterations based on Bregman divergence. This is the first algorithm that can compute the exact optimal transport plan in approximately $O(n^2)$ complexity, which largely benefits the downstream tasks.

We then apply the algorithm to generative adversarial networks and color transferring to demonstrate the efficiency of the algorithm.

Computation – Continuous Case. In some applications, OT aligns two *continuous* distributions. For example, in domain adaptation, the datasets are treated as realizations of continuous distributions. In the continuous case, (Equation 1.1) is infinite dimensional and

generally intractable. Therefore, existing literature [2] has resorted to discretize the support using a refined grid, and cast (Equation 1.1) into a finite dimensional linear programming problem. However, for complex distributions in high dimensions (e.g., images in domain adaptation), the grid size often needs to be exponentially large (e.g., exponential in dimension) to ensure a small approximation error.

To address the scalability and efficiency issues, my collaborators and I propose a new implicit generative learning-based framework for solving optimal transport problems. Specifically, we approximate γ^* by a generative model, which maps from some latent variable Z to (X, Y). For simplicity, we denote $G(Z) = [G_X(Z); G_Y(Z)] = [X; Y]$, where Z follows some simple latent distribution and G is some operator, parametrized by a deep neural network. Accordingly, instead of directly estimating the probability density of γ^*, we estimate the mapping G between Z and (X, Y) by solving

$$G^* = \operatorname*{argmin}_G \mathbb{E}_{Z\sim\rho}[c(G_X(Z), G_Y(Z))], \quad \text{subject to} \quad G_X(Z) \sim \mu,\ G_Y(Z) \sim \nu. \tag{1.3}$$

We then cast (Equation 1.3) into a minimax optimization problem using the Lagrangian multiplier method, where the Lagrangian multipliers are also approximated by deep neural networks. This eventually delivers a finite dimensional generative learning problem.

We then apply the framework to domain adaptation, and achieve the state of the art performance.

Application – Differentiable Programming. Many applications of machine learning benefit from the end-to-end differentiability, since it enables the possibility to train compositional models by gradient descent. Yet, there remain many operations in which discrete decisions are required at intermediate steps of a data processing pipeline, notably those involving sequences of discrete objects. Here, we target one of these operations – the top-k operation, i.e., finding the k largest or smallest elements from a set, which is widely used for predictive modeling in information retrieval, machine learning, and data mining.

In this work, we propose the SOFT (Scalable Optimal transport-based diFferenTiable) top-k operation as a differentiable approximation of the standard top-k operation. Specifically, motivated by the implicit differentiation techniques, we first parameterize the top-k operation in terms of the optimal solution of an OT problem. We then rule out the discontinuity by imposing entropy regularization to the optimal transport problem, and show that such a problem yields a differentiable approximation to the top-k operation.

We apply SOFT top-k operation to kNN for classification, beam search, and learning sparse attention for neural machine translation. The experimental results demonstrate significant performance gain over competing methods.

We then consider a variant of regression problem, where the correspondence between input and output data is not available. Such shuffled data is commonly observed in many real world problems. Taking flow cytometry as an example, the measuring instruments may not be able to maintain the correspondence between the samples and the measurements. Due to the combinatorial nature of the problem, most existing methods are only applicable when the sample size is small, and limited to linear regression models. To overcome such bottlenecks, we propose a new computational framework – ROBOT – for the shuffled regression problem, which is applicable to large data and complex nonlinear models. Specifically, we reformulate the regression without correspondence as a continuous optimization problem. Then by exploiting the interaction between the regression model and the data correspondence, we develop a hypergradient approach based on differentiable programming techniques. Such a hypergradient approach essentially views the data correspondence as an operator of the regression, and therefore allows us to find a better descent direction for the model parameter by differentiating through the data correspondence. ROBOT can be further extended to the inexact correspondence setting, where there may not be an exact alignment between the input and output data. Thorough numerical experiments show that ROBOT achieves better performance than existing methods in both linear and nonlinear tasks, including real-world applications such as flow cytometry and multi-object tracking.

CHAPTER 2

COMPUTATION - DISCRETE CASE

2.1 Introduction

In this chapter we focus on Wasserstein distance for discrete distributions the computation of which amounts to solving the following discrete *optimal transport* (OT) problem,

$$W(\boldsymbol{\mu}, \boldsymbol{\nu}) = \min_{\boldsymbol{\Gamma} \in \Sigma(\boldsymbol{\mu}, \boldsymbol{\nu})} \langle \boldsymbol{C}, \boldsymbol{\Gamma} \rangle. \tag{2.1}$$

Here $\boldsymbol{\mu}, \boldsymbol{\nu}$ are two probability vectors, $W(\boldsymbol{\mu}, \boldsymbol{\nu})$ is the Wasserstein distance between $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$. Matrix $\boldsymbol{C} = [c_{ij}] \in \mathbb{R}_+^{n \times n}$ is the cost matrix, whose element c_{ij} represents the distance between the i-th support point of $\boldsymbol{\mu}$ and the j-th one of $\boldsymbol{\nu}$. Notation $\langle \cdot, \cdot \rangle$ represents the Frobenius dot-product and $\Sigma(\boldsymbol{\mu}, \boldsymbol{\nu}) = \{\boldsymbol{\Gamma} \in \mathbb{R}_+^{n \times n} : \boldsymbol{\Gamma} \mathbf{1}_n = \boldsymbol{\mu}, \boldsymbol{\Gamma}^\top \mathbf{1}_n = \boldsymbol{\nu}\}$, where $\mathbf{1}_n$ represents n-dimensional vector of ones. This is a linear programming problem with typical super $O(n^3)$ complexity.

An effort by Cuturi to reduce the complexity leads to a regularized variation of (Equation 2.1) giving rise the so-called Sinkhorn distance [2]. It aims to solve an entropy regularized optimal transport problem

$$W_\epsilon(\boldsymbol{\mu}, \boldsymbol{\nu}) = \min_{\boldsymbol{\Gamma} \in \Sigma(\boldsymbol{\mu}, \boldsymbol{\nu})} \langle \boldsymbol{C}, \boldsymbol{\Gamma} \rangle + \epsilon h(\boldsymbol{\Gamma}). \tag{2.2}$$

The entropic regularizer $h(\boldsymbol{\Gamma}) = \sum_{i,j} \Gamma_{ij} \ln \Gamma_{ij}$ results in an optimization problem (Equation 2.2) that can be solved efficiently by iterative Bregman projections [3],

$$\boldsymbol{a}^{(l+1)} = \frac{\boldsymbol{\mu}}{\boldsymbol{G}\boldsymbol{b}^{(l)}}, \quad \boldsymbol{b}^{(l+1)} = \frac{\boldsymbol{\nu}}{\boldsymbol{G}^T \boldsymbol{a}^{(l+1)}}$$

starting from $b^{(0)} = \frac{1}{n}\mathbf{1}_n$, where $\boldsymbol{G} = [G_{ij}]$ and $G_{ij} = e^{-C_{ij}/\epsilon}$. The optimal solution $\mathbf{\Gamma}^*$ then takes the form $\Gamma^*_{ij} = a_i G_{ij} b_j$. The iteration is also referred as *Sinkhorn iteration*, and the method is referred as *Sinkhorn algorithm* which, recently, is proven to achieve a near-$O(n^2)$ complexity [4].

The choice of ϵ cannot be arbitrarily small. Firstly, $G_{ij} = e^{-C_{ij}/\epsilon}$ tends to underflow if ϵ is very small. The methods in [3, 5, 6] try to address this numerical instability by performing the computation in log-space, but they require a significant amount of extra exponential and logarithmic operations, and thus, compromise the advantage of efficiency. More significantly, even with the benefits of log-space computation, the linear convergence rate of the Sinkhorn algorithm is determined by the *contraction ratio* $\kappa(\boldsymbol{G})$, which approaches 1 as $\epsilon \to 0$ [7]. Consequently, we observe drastically increased number of iterations for Sinkhorn method when using small ϵ.

Can we just employ Sinkhorn distance with a moderately sized ϵ for machine learning problems so that we can get the benefits of the reduced complexity? Some applications show Sinkhorn distance can generate good results with a moderately sized ϵ [8, 9]. However, we show that in several important problems such as generative model learning and Wasserstein barycenter computation, a moderately sized ϵ will significantly degrade the performance while the Sinkhorn algorithm with a very small ϵ becomes prohibitively expensive (also shown in [10]).

In this paper, we propose a new framework, Inexact Proximal point method for Optimal Transport (IPOT) to compute the Wasserstein distance using generalized proximal point iterations based on Bregman divergence. To enhance efficiency, the proximal operator is inexactly evaluated at each iteration using projections to the probability simplex, leading to an **inexact** update at each iteration yet converging to the **exact** optimal transport solution.

Regarding the theoretical analysis of IPOT, we provide conditions on the number of inner iterations that will guarantee the linear convergence of IPOT. In fact, empirically, IPOT behaves better than the analysis: the algorithm seems to be linearly convergent with

just one inner iteration, demonstrating its efficiency. We also perform several other tests to show the excellent performance of IPOT. As we will discussed in Section subsection 2.4.2, the computation complexity is almost indistinguishable comparing to the Sinkhorn method. Yet again, IPOT avoids the lengthy and experience-based tuning of the ϵ and can converges to the true optimal transport solution robustly with respect to its own parameters. This is unquestionably important in applications where the exact sparse transportation plan is preferred. In applications where only Wasserstein distance is needed, the bias caused by regularization might also be problematic. As an example, when applying Sinkhorn to generative model learning, it causes the shrinkage of the learned distribution towards the mean, and therefore cannot cover the whole support of the target distribution adequately.

Furthermore, we develop another new algorithm based on the proposed IPOT to compute Wasserstein barycenter (see Section section 2.5). Better performance is obtained with much sharper images. It turns out that the inexact evaluation of the proximal operator blends well with Sinkhorn-like barycenter iteration.

2.2 Preliminaries

2.2.1 Wasserstein Distance and Optimal Transport

Wasserstein distance is a metric for two probability measures. Given two distributions μ and ν, the p-Wasserstein distance between them is defined as

$$W_p(\mu,\nu) := \left\{ \inf_{\gamma\in\Sigma(\mu,\nu)} \int_{\mathcal{M}\times\mathcal{M}} d^p(x,y)\mathrm{d}\gamma(x,y) \right\}^{\frac{1}{p}}, \tag{2.3}$$

where $\Sigma(\mu,\nu)$ is the set of joint distributions whose marginals are μ and ν, respectively. The above optimization problem is also called the Monge-Kantorovitch problem or *optimal transport* problem [11]. In the following, we focus on the 2-Wasserstein distance, and for convenience we write $W = W_2^2$.

When μ and ν both have finite supports, we can represent the distributions as vectors

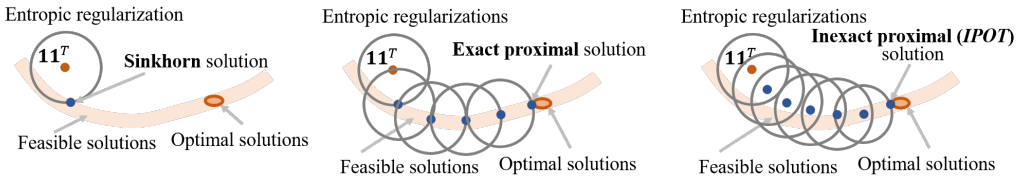


Figure 2.1: Schematic of the convergence path of (a) Sinkhorn algorithm, (b) exact proximal point algorithm and (c) inexact proximal point algorithm (IPOT). The distance shown is in Bregman sense. Sinkhorn solution is feasible and the closest to optimal solution set within the D_h constraints, but is not in the optimal solution set. However, proximal point algorithm, no matter exact or inexact, solves optimization with D_h constraints iteratively, until an optimal solution is reached.

$\boldsymbol{\mu} \in \mathbb{R}_+^{n_1}, \boldsymbol{\nu} \in \mathbb{R}_+^{n_2}$, where $\|\boldsymbol{\mu}\|_1 = \|\boldsymbol{\nu}\|_1 = 1$. Then the Wasserstein distance is computed by (Equation 2.1). In other cases, given realizations $\{x_i\}_{i=1}^{n_1}$ and $\{y_i\}_{i=1}^{n_2}$ of μ and ν, respectively, we can approximate them by empirical distributions $\widehat{\mu} = \frac{1}{n_1}\sum_{x_i} \delta_{x_i}$ and $\widehat{\nu} = \frac{1}{n_2}\sum_{y_i} \delta_{y_i}$. The supports of $\widehat{\mu}$ and $\widehat{\nu}$ are finite, so similarly we have $\boldsymbol{\mu} = \frac{1}{n_1}\mathbf{1}_{\{x_i\}}$, $\boldsymbol{\nu} = \frac{1}{n_2}\mathbf{1}_{\{y_i\}}$, and $\boldsymbol{C} = [c(x_i, y_j)] \in \mathbb{R}_+^{n_1 \times n_2}$.

The optimization problem (Equation 2.1) is a linear programming (LP) problem. LP tends to provide a sparse solution, which is preferable in applications like histogram calibration or color transferring [12]. However, the cost of LP scales at least $O(n^3 \log n)$ for general metric, where n is the number of data points [13]. As aforementioned, an alternative optimization method is the Sinkhorn algorithm in [2]. Following the same strategy, many variants of the Sinkhorn algorithm have been proposed [4, 14, 15]. Unfortunately, all these methods only approximate original optimal transport by its regularized version and their performance both in terms of numerical stability and computational complexity is very sensitive to the choice of ϵ.

2.2.2 Proximal Point Method

Proximal point methods are widely used in optimization [16, 17, 18, 19]. Given a convex objective function f defined on $\mathcal{X}$ with optimal solution set $\mathcal{X}^* \subset \mathcal{X}$, proximal point

algorithm aims to solve

$$\text{argmin}_{x\in\mathcal{X}} f(x). \tag{2.4}$$

In order to solve Problem (Equation 2.4), the algorithm generates a sequence $\{x^{(t)}\}_{t=1,2,...}$ by the following generalized proximal point iterations:

$$x^{(t+1)} = \text{argmin}_{x\in\mathcal{X}} f(x) + \beta^{(t)} d(x, x^{(t)}), \tag{2.5}$$

where d is a regularization term used to define the proximal operator, usually defined to be a closed proper convex function. Commonly, d adopts the square of Euclidean distance, i.e., $d(x, y) = \|x - y\|_2^2$. When Euclidean distance is used, the sequence $\{x^{(t)}\}$ converges to an element in $\mathcal{X}^*$ almost surely.

The proximal point method has many advantages, e.g, it has a robust convergence behavior — a fairly mild condition on β guarantee its convergence and the specific choice of β generally just affects its convergence rate. Moreover, even if the proximal operator defined in (Equation 2.5) is not exactly evaluated in each iteration, giving rise to inexact proximal point methods, the global convergence of which with local linear rate is still guaranteed under certain conditions [20, 21].

2.3 Bregman Divergence Based Proximal Point Method

2.3.1 Proposed Method

Our key idea is to use Bregman divergence D_h as the regularization in evaluating the proximal operator in (Equation 2.5), i.e.

$$\mathbf{\Gamma}^{(t+1)} = \text{argmin}_{\mathbf{\Gamma}\in\Sigma(\boldsymbol{\mu},\boldsymbol{\nu})} \langle \boldsymbol{C}, \mathbf{\Gamma}\rangle + \beta^{(t)} D_h(\mathbf{\Gamma}, \mathbf{\Gamma}^{(t)}), \tag{2.6}$$

where Bregman divergence D_h based on entropy function $h(x) = \sum_i x_i \ln x_i$ takes the form (see supplementary material for more)

$$D_h(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} x_i \log \frac{x_i}{y_i} - \sum_{i=1}^{n} x_i + \sum_{i=1}^{n} y_i. \tag{2.7}$$

Substituting Bregman divergence into proximal point iteration (Equation 2.6), with simplex constraints, we obtain

$$\boldsymbol{\Gamma}^{(t+1)} = \operatorname{argmin}_{\boldsymbol{\Gamma} \in \Sigma(\boldsymbol{\mu}, \boldsymbol{\nu})} \langle \boldsymbol{C} - \beta^{(t)} \log \boldsymbol{\Gamma}^{(t)}, \boldsymbol{\Gamma} \rangle + \beta^{(t)} h(\boldsymbol{\Gamma}). \tag{2.8}$$

Algorithm 1 IPOT($\boldsymbol{\mu}, \boldsymbol{\nu}, \boldsymbol{C}$)

Input: Probabilities $\{\boldsymbol{\mu}, \boldsymbol{\nu}\}$ on support points $\{x_i\}_{i=1}^m$, $\{y_j\}_{j=1}^n$, metric matrix $\boldsymbol{C} = [\|x_i - y_j\|]$

$\boldsymbol{b} \leftarrow \frac{1}{m}\mathbf{1}_m$

$G_{ij} \leftarrow e^{-\frac{C_{ij}}{\beta}}$

$\boldsymbol{\Gamma}^{(1)} \leftarrow \mathbf{1}\mathbf{1}^T$

for $t = 1, 2, 3, ...$ **do**

  $\boldsymbol{Q} \leftarrow \boldsymbol{G} \odot \boldsymbol{\Gamma}^{(t)}$

  for $l = 1, 2, 3, ..., L$ **do** // Usually set $L = 1$

    $\boldsymbol{a} \leftarrow \frac{\boldsymbol{\mu}}{\boldsymbol{Q}\boldsymbol{b}}$, $\boldsymbol{b} \leftarrow \frac{\boldsymbol{\nu}}{\boldsymbol{Q}^T\boldsymbol{a}}$

  end for

  $\boldsymbol{\Gamma}^{(t+1)} \leftarrow \text{diag}(\boldsymbol{a})\boldsymbol{Q}\text{diag}(\boldsymbol{b})$

end for

Denote $\boldsymbol{C}' = \boldsymbol{C} - \beta^{(t)} \ln \boldsymbol{\Gamma}^{(t)}$. The optimization (Equation 2.8) can be solved by Sinkhorn iteration by replacing G_{ij} by $G'_{ij} = e^{-C'_{ij}/\beta^{(t)}} = \Gamma_{ij}^{(t)} e^{-C_{ij}/\beta^{(t)}}$. As we will later shown in Section subsection 2.3.2, as $t \to \infty$, $\boldsymbol{\Gamma}^{(t)}$ will converge to an optimal transportation plan.

Figure Figure 2.1 illustrates how Sinkhorn and IPOT solutions approach optimal solution in sense of Bregman divergence. First, let's consider Sinkhorn algorithm. The loss function of Sinkhorn has regularization term $\epsilon h(\mathbf{\Gamma})$, which can be rewritten as constraint $D_h(\mathbf{\Gamma}, \mathbf{1}\mathbf{1}^T) \leq \eta$ for some $\eta > 0$. So in the left figure Sinkhorn solution is feasible within the D_h constraints and the closest to optimal solution set. Proximal point algorithm, on the other hand, solves optimization with D_h constraints iteratively, until an optimal solution is reached. Exact proximal point method is when (Equation 2.8) solved exactly. It provides a feasible solution that is closest to the optimal solution set in each step, and finally reach an optimal solution. Inexact proximal point method is when (Equation 2.8) is solved inexactly. Since we use Sinkhorn iteration to solve (Equation 2.8), this is when the iteration number of the inner optimization is not enough to converge. In each step, the solution might not be feasible or closest to the optimal set, but eventually it converges to an optimal solution.

The algorithm is shown in Algorithm Algorithm 1. For simplicity we use $\beta = \beta^{(t)}$. Denote $\text{diag}(\boldsymbol{a})$ the diagonal matrix with a_i as its ith diagonal elements. Denote $\odot$ as element-wise matrix multiplication and $\frac{(\cdot)}{(\cdot)}$ as element-wise division. We use warm start to improve the efficiency, i.e. in each proximal point iteration, we use the final value of $\boldsymbol{a}$ and $\boldsymbol{b}$ from last proximal point iteration as initialization instead of $\boldsymbol{b}^{(0)} = \mathbf{1}_m$. Later we will show empirically IPOT will converge under a large range of β with $L = 1$, a single inner iteration will suffice.

2.3.2 Theoretical Analysis

Classical proximal point algorithm has sublinear convergence rate. However, combining Bregman distance and simplex constraints, we prove stronger convergence rate - a linear rate. First, we consider when the optimization problem (Equation 2.8) is solved exactly, we have a linear convergence rate guaranteed by the following theorem.

Theorem 1. Let $\{x^{(t)}\}$ be a sequence generated by the proximal point algorithm

$$x^{(t+1)} = \text{argmin}_{x\in\mathcal{X}} f(x) + \beta^{(t)} D_h(x, x^{(t)}),$$

where f is continuous and convex. Assume $f^* = \min f(x) > -\infty$. Then, with $\sum_{t=0}^{\infty} \beta^{(t)} = \infty$, we have

$$f(x^{(t)}) \downarrow f^*.$$

If we further assume f is linear and $\mathcal{X}$ is bounded, the algorithm has linear convergence rate.

More importantly, the following theorem gives us a guarantee of convergence when (Equation 2.8) is solved inexactly.

Theorem 2. Let $\{x^{(t)}\}$ be the sequence generated by the Bregman distance based proximal point algorithm with inexact scheme (i.e., finite number of inner iterations are employed). Define an error sequence $\{e^{(t)}\}$ where

$$e^{(t+1)} \in \beta^{(t)} \left[\nabla f(x^{(t+1)}) + \partial \iota_{\mathcal{X}}(x^{(t+1)})\right] + \left[\nabla h(x^{(t+1)}) - \nabla h(x^{(t)})\right],$$

where $\iota_{\mathcal{X}}$ is the indicator function of set $\mathcal{X}$. If the sequence $\{e^k\}$ satisfies $\sum_{k=1}^{\infty} \|e^k\| < \infty$ and $\sum_{k=1}^{\infty} \langle e^k, x^{(t)}\rangle$ exists and is finite, then $\{x^{(t)}\}$ converges to x^{∞} with $f(x^{\infty}) = f^*$. If the sequence $\{e^{(t)}\}$ satisfies that exist $\rho \in (0, 1)$ such that$\|e^{(t)}\| \leq \rho^t$, $\langle e^{(t)}, x^{(t)}\rangle \leq \rho^t$ and with assumptions that f is linear and $\mathcal{X}$ is bounded, then $\{x^{(t)}\}$ converges linearly.

The proof of both theorems is given in the supplementary material. Theorem Theorem 2 guarantees the convergence of inexact proximal point method — as long as L satisfies the given conditions, the IPOT algorithm would converge linearly.

Now we know IPOT can converge to the exact Wasserstein distance. What if an entropic regularization is wanted? Please refer to the suplement material for how IPOT can achieve regularizations with early stopping.

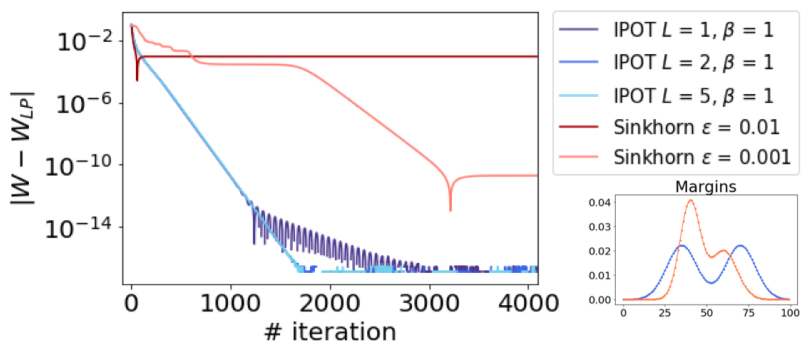


Figure 2.2: The plot of differences in computed Wasserstein distances w.r.t. number of iterations. Here, W are the Wasserstein distance computed at current iteration. W_{LP} is computed by simplex method, and is used as ground truth. The test adopts $c(x, y) = ||x - y||_2$. The right lower figure is the two input margins for the test.

2.4 Empirical Analysis

In this section we will illustrate the convergence behavior with respect to inner iteration number L, the scalability of IPOT, and the issue with entropy regularization. We leverage the implementation of Sinkhorn iteration and LP solver based on Python package POT [22], and use Pytorch to parallel some of the implementation.

2.4.1 Convergence Rate

A simple illustration task of calculating the Wasserstein distance of two 1D distribution (shown in the bottom right corner of Figure Figure 2.2) is conducted as numerical validation of the convergence theorems proved in Section subsection 2.3.2. To be clear, the use of two 1D distribution is only for visualization purpose. We also did tests on empirical distribution of 64D Gaussian distributed data, and the result shows the same trend. We include more discussion in the supplement material.

Figure Figure 2.2 shows the convergence of IPOT under different L. The algorithm has empirically linear convergence rate even under very small L. Thus, for simplicity, we use $L = 1$ for later tests.

Furthermore, as illustrated in Figure Figure 2.1, the proposed IPOT method converges

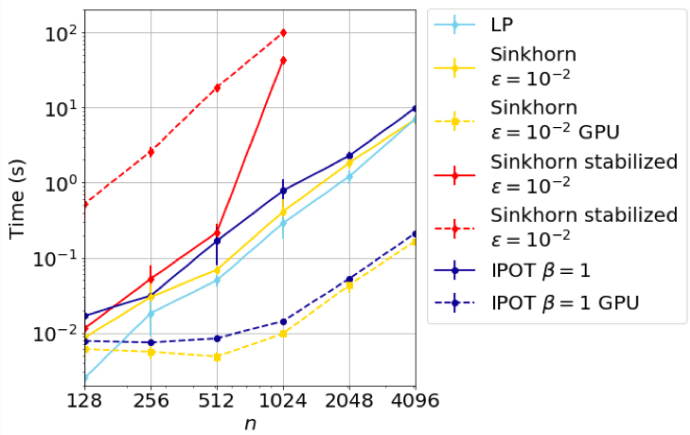


Figure 2.3: Log-log plot of average time used to achieve 1e-4 relative precision with error bar. Each point is obtained by the average of 6 tests on different datasets.

to the real Wasserstein distance even with large β, while the Sinkhorn method has distinct bias with much smaller ϵ.

2.4.2 Scalability

We conduct the following scalability test to show the computation time of the proposed IPOT comparing to the state-of-art benchmarks. The optimal transport problem is conducted between the two empirical distributions of 16D uniformly distributed data (See Section subsection 2.2.1 for formulation). Besides proposed IPOT algorithm (see Algorithm Algorithm 1 with $L = 1$), the Sinkhorn algorithm follows [2] and the stabilized Sinkhorn algorithm follows [5]. The result of the scalability test is shown in Figure Figure 2.3. The LP solver has a good performance under the current experiment settings. But LP solver is not guaranteed to have approximately $O(n^2)$ convergence as shown here. Moreover, LP method is difficult to parallel. Readers who are interested please refer to experiments in [2].

Sinkhorn and IPOT can be paralleled conveniently, so we provide both CPU and GPU tests here. Under this setting, IPOT takes approximately the same resources as Sinkhorn at $\epsilon = 0.01$. For smaller ϵ, original Sinkhorn will underflow, and we need to use stabilized Sinkhorn. Stabilized Sinkhorn is much more expensive than IPOT, especially for large datasets and small ϵ.

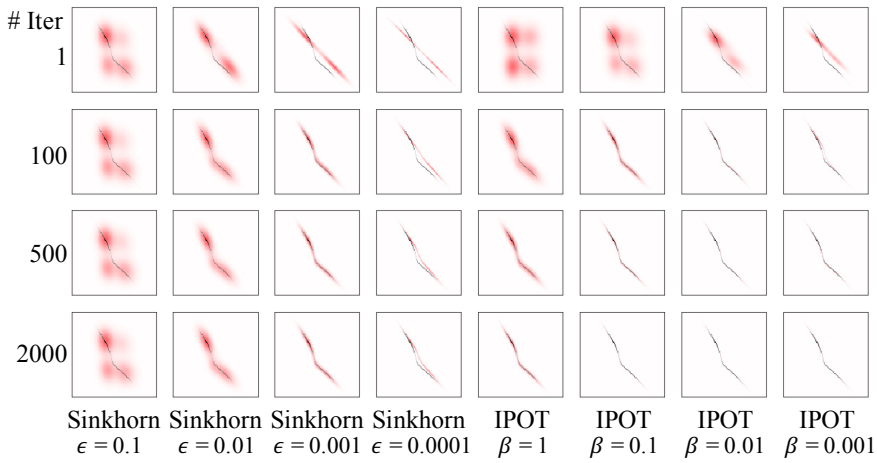


Figure 2.4: The transportation plan generated by Sinkhorn and IPOT methods at different iteration number. The red colormap is the result from Sinkhorn or IPOT method, while the black wire is the result of simplex method for comparison. In the right lower plans, the red and the black is almost identical.

Note that we also try to use the method proposed in [23] for ϵ scaling, to help the convergence when $\epsilon \to 0$. However, although it is faster than Sinkhorn method when data size is smaller than 1024, the time used at 1024 is already around 2×10^3s. Therefore we didn't include this method in the figure.

2.4.3 Effect of Entropy Regularization

We have shown that IPOT can converge to exact Wasserstein distance with complexity comparable to Sinkhorn (see Figure Figure 2.1 and Figure 2.3) and as we claimed in Section section 2.1 this is important in some of the learning problems.

But in what cases is the exact Wasserstein distance truly needed? How will the entropy regularization term affect the result in different applications? In this section, we will discuss the exact transportation plan with sparsity preference and the advantage of exact Wasserstein distance in learning the generative models.

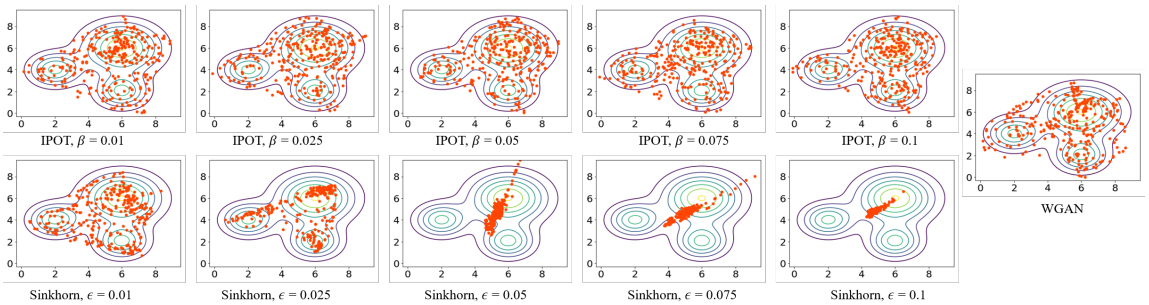


Figure 2.5: The sequences of learning results using IPOT, Sinkhorn, and original WGAN. In each figure, the orange dots are samples of generated data, while the contour represents the ground truth distribution.

Sparsity of the Transportation Plan

In applications such as histogram calibration and color transferring, an exact and sparse transportation plan is wanted. In this section we conduct tests on the sparsity of the transportation plan using the two distributions shown in Figure Figure 2.2 for both IPOT and Sinkhorn methods with different regularization coefficients. Figure Figure 2.4 visualize the different transportation plans. The red colormap is the result from Sinkhorn or IPOT method, where the black wire beneath is the result by simplex method as ground truth. To be clear, the different number of interaction of IPOT means the number of the outer iteration with still $L = 1$ inner iteration.

The proposed IPOT method can always converge to the sparse ground truth with enough iteration and it is very robust with respect to the parameter β, i.e., there is little visual difference with β changing from 0.1 to 0.001. Furthermore, even with large $\beta = 1$, the optimal plan is still sparse and acceptable. In addition, if some smoothness is wanted, IPOT method would also be able to work with early stopping. The degree of smoothness can be easily adjusted by adjusting the number of iterations if needed.

On the other hand, the optimal plans obtained by Sinkhorn has two issues. If the ϵ is chosen to be large (i.e., $\epsilon = 0.1$ or 0.01), the optimal plan are blur i.e., neither exact nor sparse. In downstream applications, the non-sparse structure of transportation plan make it difficult to extract the transportation map from source distribution to target distri-

bution. However if the ϵ is chosen to be small (i.e., $\epsilon = 0.0001$), it needs more iterations to converge. For example, the Sinkhorn $\epsilon = 0.0001$ case still cannot converge after 2000 iterations. So in Sinkhorn applications, ϵ needs to be selected carefully. This fine tuning issue can be avoid by the proposed IPOT method, since IPOT is robust to the parameter β.

Shrinkage Problem in Learning Generative Models

As shown in Equation (Equation 2.2), Sinkhorn method use entropy to penalize the optimization target and has biased evaluation of Wasserstein distance. The inaccuracy will affect the performance of the learning problem where Wasserstein metric is served as loss function.

In order to better illustrate the affect of the inaccurate Wasserstein distance, we consider the task of learning generative models, specifically, Wasserstein GAN [24]. Similar to other GAN, WGAN seeks to learn a generated distrubution to approximate a target distribution, except using Wasserstein distance as the loss that measures the distance between the generated distribution and target distribution. It uses the Kantorovitch dual formulation to compute Wasserstein distance.

In this section, we train a Wasserstein GAN with the dual formulation substituted by Sinkhorn and IPOT methods. Detailed derivation can be found in supplementary material. Meanwhile, the standard approach of using dual form proposed in [24] is also compared. Note that the purpose of this section is not to propose a new GAN but visualize how proposed IPOT can avoid the possible negative influence introduced by the inaccuracy of the entropy regularization in the Sinkhorn method.

We claim that result of Sinkhorn method with moderate size ϵ tends to shrink towards the mean, so the learned distribution cannot cover all the support of target distribution. To demonstrate the reason of this trend, consider the extreme condition when $\epsilon \to \infty$, the loss

function becomes

$$\mathbf{\Gamma}^* = \underset{\mathbf{\Gamma}}{\operatorname{argmin}}\, h(\mathbf{\Gamma}) = \underset{\mathbf{\Gamma}}{\operatorname{argmin}}\, D_h(\mathbf{\Gamma}, \mathbf{1}\mathbf{1}^T/n^2).$$

So $\mathbf{\Gamma}^* = \mathbf{1}\mathbf{1}^T/n^2$. If we view $\{x_i\}$ and $\{y_j : y_j = g_\theta(z_j)\}$ as the realizations of random variables X and Y, the optimal Sinkhorn distance W_ϵ is expected to be

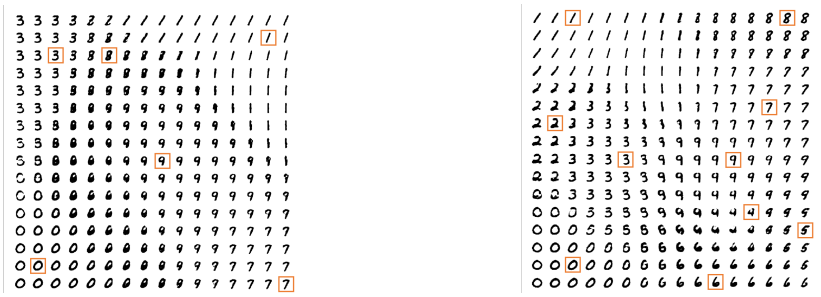
$$\begin{aligned}\mathbb{E}_{X,Y}[W_\epsilon] &= \mathbb{E}_{X,Y}[\langle \mathbf{\Gamma}^*, \boldsymbol{C}\rangle] \\ &= \mathbb{E}_{X,Y}[\sum_{i,j} ||x_i - y_j||_2^2] = n^2(\text{Var}(X) + (\overline{X} - \overline{Y})^2 + \text{Var}(Y)),\end{aligned}$$

where n is the data size, $\overline{(\cdot)}$ is the mean of random variable, and $\text{Var}(\cdot)$ is the variance. At the minimum of the distance, the mean of generated data $\{y_j\}$ is the same as $\{x_i\}$, but the variance is zero. Therefore, the learned distribution would shrink asymptotically toward the data mean due to smoothing the effect of regularization.

However, the proposed IPOT method is free from the above shrinking issue since the exact Wasserstein distance can be found with the approximately the same cost (see Section subsection 2.4.1 and Section subsection 2.4.2). Now we illustrate the shrinkage problem by the following experiments.

Experiments on 2D Synthetic Data First, we conduct a 2D toy example to demonstrate the affect of regularization. We use a 2D-2D NN as generator to learn a mapping from uniformly distributed noise to mixture of Gaussian distributed real data. Since as shown in Figure Figure 2.2, Sinkhorn may need more iteration to converge, in this experiment, IPOT uses 200 iterations and Sinkhorn uses 500 iterations.

Figure Figure 2.5 shows the results. As ϵ varies from 0.01 to 0.1, the learned distribution of Sinkhorn gradually shrinks to the mean of target distribution, again this is because the inaccurcy in calculating the Wasserstein distance. On the contrary, since IPOT can converge to the exact Wasserstein distance regardless of different β, the result robustly cover the



(a) Sinkhorn $\epsilon = 1$: digits 0,1,3,7,8,9 are covered.

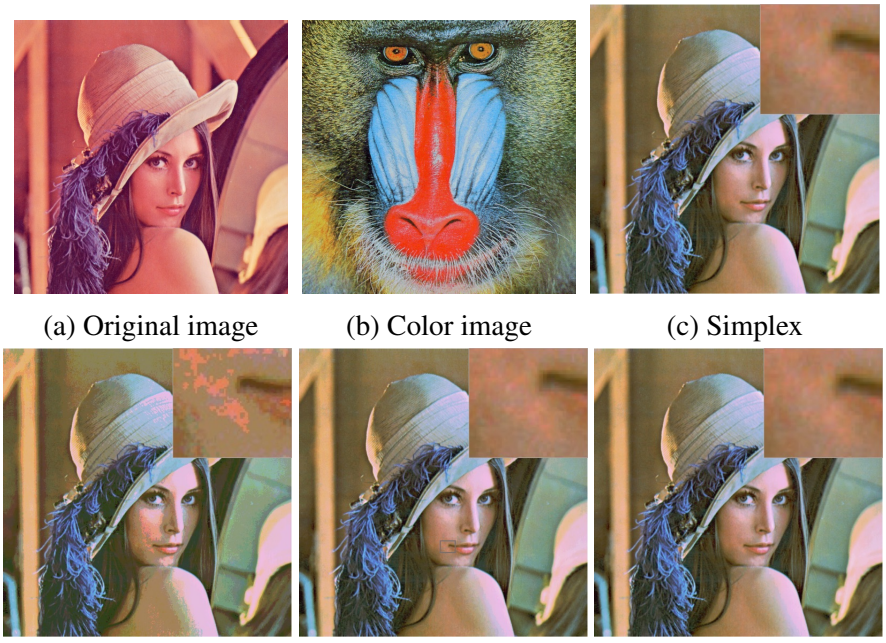
(b) IPOT $\beta = 1$: all digits are covered.

Figure 2.6: Plots of MNIST learning result under comparable resources. They both use batch size=200, number of hidden layer=1, number of nodes of hidden layer=500, number of iteration=200, learning rate = 1e-4.

whole support of target distribution. Furthermore, comparing to the dual form method used in the WGAN, the proposed IPOT method is better in small scale cases and can achieve similar performance in large scale cases [8]. This is mainly because the discriminator neural network used in WGAN is susceptible to overfitting in low dimensional cases, and it exceeds the objective of this paper.

Experiments on Higher Dimensional Data For higher dimensional data, we cannot visualize the final generated distribution as done in the 2D test. So in order to demonstrate IPOT has little shrinkage issue, we set the latent space to be 2D, and visualize it by plotting the images generated at dense grid points on the latent space. Due to the low dimensional latent space, We perform the experiment using MNIST dataset. Note that this is mainly for the convenience in visualization, the whole shrinkage-free property of IPOT method is also extendable to more complex learning problems. Associated with the MNIST dataset, we use a generator $g_\theta : \mathbb{R}^2 \mapsto \mathbb{R}^{784}$, noise data $\{z_j\} \sim \text{Unif}([0,1]^2)$ as input, and one fully connected hidden layer with 500 nodes.

Figure Figure 2.6 shows an example of generated results. The Sinkhorn results look authentic, but we can only find some of the digits in it. This is exactly the consequence of shrinkage due to the inaccurate calculation of Wasserstein distance - in the domain where the density of learned distribution is nonzero, the density of target distribution is usually



(a) Original image (b) Color image (c) Simplex

(d) Sinkhorn ϵ=10^{-2} (e) Sinkhorn ϵ=10^{-3} (f) IPOT

Figure 2.7: An example of color transferring. The right upper corner of each generated image shows the zoom-in of the color detail of the mouth corner.

nonzero; but in some part of the domain where the density of target distribution is nonzero, the learned distribution is zero. In the example of Figure Figure 2.6 (a), the learned distribution cannot cover the support of digits 2,4,5,6 while when using IPOT to calculate the Wasserstein loss, all ten digits are can be recovered in Figure 2.6 (b), which shows the coverage of the whole domain of the target distribution. In supplementary material we provide more examples, e.g., if a larger ϵ is used, Sinkhorn generator would shrink to one point, and hence cannot learn anything, while the IPOT method is robust to its parameter β and covers more digits.

2.4.4 Color Transferring

Optimal transport is directly applicable to many applications, such as color transferring and histogram calibration. We will show the result of color transferring and why accurate transportation map is superior to entropically regularized ones.

The goal of color transferring is to transfer the tonality of a target image into a source

image. This is usually done by imposing the histogram of the color palette of one image to another image. Since Reinhard et al. [25], many methods [12, 26] are developed to do so by learning the transformation between the two histograms. Experiments in [27] have shown that transformation based on optimal transport map outperforms state-of-the-art techniques for challenging images.

Same as other prime-form Wasserstein distance solvers [13, 2], the proximal point method provide a transportation map. By definition, the map is a transportation from the source distribution to a target one with minimum cost. Therefore it provides a way to transform a histogram to another.

One example is shown in figure Figure 2.7. We use three different maps to transform the RGB channels, respectively. For each channel, there are at most 256 bins. Therefore, using three channels separately is more efficient than treating the colors as 3D data. Figure Figure 2.7 shows proximal point method can produce identical result as linear programming at convergence, while the results produced by Sinkhorn method differ w.r.t. ϵ.

2.5 Wasserstein Barycenter by IPOT

Wasserstein barycenter is widely used in machine learning and computer vision due to its nice property [3, 28]. Given a set of distributions $\mathcal{P} = \{\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_K\}$, their Wasserstein barycenter is defined as

$$\boldsymbol{q}^*(\mathcal{P}, \boldsymbol{\lambda}) = \operatorname{argmin}_{\boldsymbol{q} \in \mathcal{Q}} \sum_{k=1}^{K} \lambda_k W(\boldsymbol{q}, \boldsymbol{p}_k) \tag{2.9}$$

where W is the Wasserstein distance, and $\mathcal{Q}$ is in the space of probability distributions, and $\sum_{k=1}^{K} \lambda_k = 1$.

Algorithm 2 computing Wasserstein barycenter

Input: The probability vector set $\{\boldsymbol{p}_k\}$ on grid $\{y_i\}_{i=1}^n$
$\boldsymbol{b}_k \leftarrow \frac{1}{n}\mathbf{1}_n, \forall k = 1, 2, ..., K$
$C_{ij} \leftarrow c(y_i, y_j) := ||y_i - y_j||_2^2$
$G_{ij} \leftarrow e^{-\frac{C_{ij}}{\beta}}$
$\boldsymbol{\Gamma}_k \leftarrow \mathbf{1}\mathbf{1}^T$
for $t = 1, 2, 3, ...$ **do**
  $\boldsymbol{H}_k \leftarrow \boldsymbol{G} \odot \boldsymbol{\Gamma}_k, \forall k = 1, 2, ..., K$
  for $l = 1, 2, 3, ..., L$ **do**
    $\boldsymbol{a}_k \leftarrow \frac{\boldsymbol{q}}{\boldsymbol{H}_k \boldsymbol{b}_k}, \forall k = 1, 2, ..., K,$
    $\boldsymbol{b}_k \leftarrow \frac{\boldsymbol{p}_k}{\boldsymbol{H}_k^T \boldsymbol{a}_k}, \forall k = 1, 2, ..., K$
    $\boldsymbol{q} \leftarrow \prod_{k=1}^K (\boldsymbol{a}_k \odot (\boldsymbol{H}_k \boldsymbol{b}_k))^{\lambda_k}$
  end for
  $\boldsymbol{\Gamma}_k \leftarrow \text{diag}(\boldsymbol{a}_k)\boldsymbol{H}_k\text{diag}(\boldsymbol{b}_k), \forall k = 1, 2, ..., K$
end for
Return $\boldsymbol{q}$

2.5.1 IPOT-like Algorithm

The idea of our IPOT method can be generalized to learn Wasserstein barycenter. In particular, plugging the definition of Wasserstein distance in (Equation 2.1) into (Equation 2.9) with some derivation (see supplementary material for full derivation), we get the proximal point iteration for barycenter analogous to (Equation 2.6) as

$$\{\boldsymbol{\Gamma}_k^{(t+1)}\} = \underset{\{\boldsymbol{\Gamma}_k\}}{\text{argmin}} \sum_{k=1}^K \lambda_k \{\langle \boldsymbol{\Gamma}_k, \boldsymbol{C} \rangle + \beta^{(t)} D_h(\boldsymbol{\Gamma}_k, \boldsymbol{\Gamma}_k^{(t)})\}$$
$$\text{s.t.} \quad \boldsymbol{\Gamma}_k \mathbf{1} = \boldsymbol{p}_k, \forall k, \quad \exists \boldsymbol{q}, \boldsymbol{\Gamma}_k^T \mathbf{1} = \boldsymbol{q}.$$

The minimization in each proximal step is solved by Sinkhorn barycenter iteration [3]. We provide the detailed algorithm in Algorithm Algorithm 2. The same as Algorithms Algorithm 1, this algorithm can also converge with $L = 1$ and a large range of β.

2.5.2 Learning Barycenter

We test our proximal point barycenter algorithm on MNIST dataset, borrowing the idea from [29]. Here, the images in MNIST dataset is randomly uniformly reshape to half to double of its original size, and the reshaped images have random bias towards corner. After that, the images are mapped into 50×50 grid. For each digit we use 50 of the reshaped images with the same weights as the dataset to compute the barycenter. All results are computed using 50 iterations and under $\epsilon, \beta = 0.001$. So for proximal point method, the regularization is approximately the same as $\epsilon = 2 \times 10^{-5}$, which is pretty small. We compare our method with state-of-art Sinkhorn based methods [29], [30] and [3]. Among the four methods, the convolutional method [30] is different in terms of that it only handles structural input tested here and does not require $O(n^2)$ storage, unlike other three general purpose methods.

We are also aware of that there are other literatures for Wassersetin barycenter, such as [31] and [32], but they are targeting a more complicated setting, and has a different convergence rate (i.e. sublinear rate) than the methods we provide here.

The results (Figure Figure 2.8) from proximal point algorithm are clear, while the results of Sinkhorn based algorithms suffer blurry effect due to entropic regularization.

While the time complexity of our method is in the same order of magnitude with Sinkhorn algorithm [3], the space complexity is K times of it, because K different transport maps need to be stored. This might cause pressure to memory for large K. Therefore, a sequential method is needed. We left this to future work.

2.6 Conclusion

We proposed a proximal point method - IPOT - based on Bregman distance to solve optimal transport problem. Different from the Sinkhorn method, IPOT algorithm can converge to ground truth even if the inner optimization iteration only performs once. This nice prop-

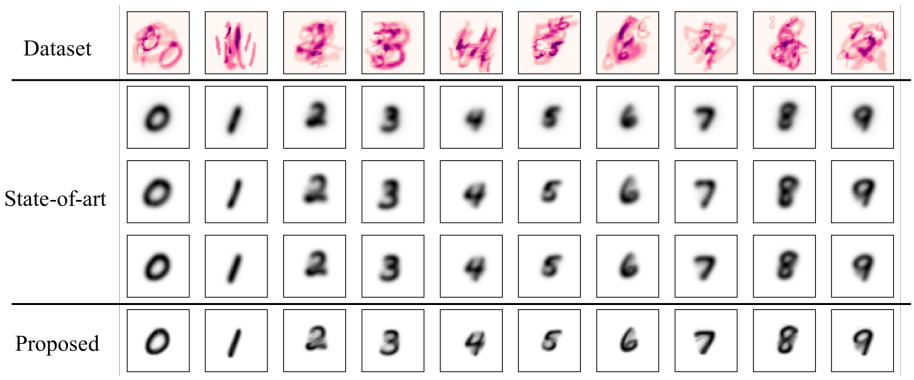


Figure 2.8: The result of barycenter. For each digit, we randomly choose 8 of 50 scaled and shifted images to demonstrate the input data. From the top to the bottom, we show (top row) the demo of input data; (second row) the results based on [29]; (third row) the result based on [30]; (fourth row) the result based on [3]; (bottom row) the results based on inexact proximal point algorithm.

erty results in similar convergence and computation time comparing to Sinkhorn method. However, IPOT provides a robust and accurate computation of Wasserstein distance and associated transportation plan, which leads to a better performance in image transformation and avoids the shrinkage in generative models. We also apply the IPOT idea to calculate the Wasserstein barycenter. The proposed method can generate much sharper results than state-of-art due to the exact computation of the Wasserstein distance.

CHAPTER 3

COMPUTATION – CONTINUOUS CASE

3.1 Introduction

In this chapter we consoder the computation of the comtinuous case OT. Specifically, we consider two sets of d-dimensional data, which are generated from two different distributions denoted by $X \sim \mu$ and $Y \sim \nu$.[1] We aim to find an optimal joint distribution γ of X and Y, which minimizes the expectation on some cost function c, i.e.,

$$\gamma^* = \underset{\gamma \in \Pi(\mu,\nu)}{\operatorname{argmin}} \mathbb{E}_{(X,Y)\sim\gamma}[c(X,Y)], \tag{3.1}$$

The constraint $\gamma \in \Pi(\mu,\nu)$ requires the marginal distribution of X and Y in γ to be identical to μ and ν, respectively. Existing literature often refers to the optimal expected cost $\mathcal{W}^*(\mu,\nu) = \mathbb{E}_{(X,Y)\sim\gamma^*}[c(X,Y)]$ as *Wasserstein distance*, and γ^* as the *optimal transport plan*. For domain adaptation, the function c measures the discrepancy between X and Y, and the optimal transport plan γ^* essentially reveals the transfer of the knowledge from source X to target Y. For resource allocation, the function c is the cost of assigning resource X to receiver Y, and the optimal transport plan γ^* essentially yields the optimal assignment.

Since (Equation 3.1) is an optimization problem over the space of distributions, the problem is infinite dimensional and generally intractable when μ and ν are continuous distributions. Therefore, existing literature has resorted to finite dimensional approximations. For example, [2] propose to discretize the support using a refined grid, and cast (Equation 3.1) into a finite dimensional linear programming problem. However, for com-

[1] The optimal transport can also handle more than two distributions. See Section section 3.3 for more details.

plex distributions in high dimensions (e.g., images in domain adaptation), the grid size often needs to be exponentially large (e.g., exponential in dimension) to ensure a small approximation error (due to discretization). Under such a regime, conventional linear programming algorithms do not scale well, e.g., the interior point method in conjunction with the Newton's method takes $\mathcal{O}(n^3 \log n)$ time, where n is the grid size. To ease such a scalability issue, [2] propose an entropy regularization-based Sinkhorn algorithm, which requires the computational cost of $\mathcal{O}(n^2)$, but still fail to scale to large problems.

While there exist several scalable stochastic algorithms for computing Wasserstein distance for continuous distributions μ and ν [33, 34, 35], they cannot compute the optimal transport plan γ^* (see Section section 3.7 for more discussion), which is crucial in the aforementioned applications.

To address the scalability and efficiency issues, we propose a new implicit generative learning-based framework for solving optimal transport problems. Specifically, we approximate γ^* by a generative model, which maps from some latent variable Z to (X, Y). For simplicity, we denote

$$\left[\frac{X}{Y}\right] = G(Z) = \left[\frac{G_X(Z)}{G_Y(Z)}\right] \quad \text{with} \quad Z \sim \rho, \tag{3.2}$$

where ρ is some simple latent distribution and G is some operator, e.g., deep neural network or neural ordinary differential equation (ODE). Accordingly, instead of directly estimating the probability density of γ^*, we estimate the mapping G between Z and (X, Y) by solving

$$G^* = \underset{G}{\operatorname{argmin}} \quad \mathbb{E}_{Z\sim\rho}[c(G_X(Z), G_Y(Z))], \quad \text{subject to} \quad G_X(Z) \sim \mu,\ G_Y(Z) \sim \nu \tag{3.3}$$

We then cast (Equation 3.3) into a minimax optimization problem using the Lagrangian multiplier method. As the constraints in (Equation 3.3) are over the space of continuous

distributions, the Lagrangian multiplier is actually infinite dimensional. Thus, we propose to approximate the Lagrangian multiplier by deep neural networks, which eventually delivers a finite dimensional generative learning problem.

Our proposed framework has three major benefits: (1) Our formulated minimax optimization problem can be efficiently solved by primal dual stochastic gradient-type algorithms. Many empirical studies have corroborated that these algorithms can easily scale to very large minimax problems in machine learning [36]; (2) Our framework can take advantage of recent advances in deep learning. Many empirical evidences have suggested that deep neural networks can effectively adapt to data with intrinsic low dimensional structures [37, 38]. Although they are often overparameterized, due to the inductive biases of the training algorithms, the intrinsic dimensions of deep neural networks are usually controlled very well, which avoids the curse of dimensionality; (3) Our adopted generative models allow us to efficiently sample from the optimal transport plan. This is very convenient for certain downstream applications such as domain adaptation, where we can generate infinitely many data points paired across domains [39].

Moreover, the proposed framework can also recover the density of entropy regularized optimal transport plan. Specifically, we adopt the neural Ordinary Differential Equation (ODE) approach in [40] to model the dynamics that how Z gradually evolves to $G(Z)$. We then derive the ODE that describes how the density evolves, and solve the density of the transport plan from the ODE. The recovery of density requires no extra parameters, and can be evaluated efficiently.

Notations: Given a matrix $A \in \mathbb{R}^{d\times d}$, $\det(A)$ denotes its determinant, $\mathrm{tr}(A) = \sum_i A_{ii}$ denotes its trace, $\|A\|_{\mathrm{F}} = \sqrt{\sum_{i,j} A_{ij}^2}$ denotes its Frobenius norm, and $|A|$ denotes a matrix with $[|A|]_{ij} = |A_{ij}|$. We use $\dim(v)$ to denote the dimension of a vector v.

3.2 Background

We briefly review some background knowledge on optimal transport and implicit generative learning.

Optimal Transport: The idea of optimal transport (OT) originally comes from [1], which proposes to solve the following problem,

$$T^* = \underset{T(X)\sim\nu}{\operatorname{argmin}} \mathbb{E}_{X\sim\mu}[c(X, T(X))], \tag{3.4}$$

where $T(\cdot)$ is a mapping from the space of μ to the space of ν. The mapping T^* is referred to as *Monge map*, and (Equation 3.4) is referred to as Monge formulation of optimal transport.

Monge formulation, however, is not necessarily feasible. For example, when X is a constant random variable and Y is not, there does not exist such a map T satisfying $T(X) \sim \nu$. The Kantorovich formulation of our interest in (Equation 3.1) is essentially a relaxation of (Equation 3.4) by replacing the deterministic mapping with the coupling between μ and ν. Consequently, *Kantorovich formulation* is guaranteed to be feasible and becomes the classical formulation of optimal transport in existing literature [3, 41, 42, 30, 43].

Implicit Generative Learning: For generative learning problems, direct estimation of a probability density function is not always convenient. For example, we may not have enough prior knowledge to specify an appropriate parametric form of the probability density function (pdf). Even when an appropriate parametric pdf is available, computing the maximum likelihood estimator (MLE) can be sometimes neither efficient nor scalable. To address these issues, we resort to implicit generative learning, which do not directly specify the density. Specifically, we consider that the observed variable X is generated by transforming a latent random variable Z (with some known distribution ρ) through some unknown mapping $G(\cdot)$, i.e., $X = G(Z)$. We then can train a generative model by estimating $G(\cdot)$ with a properly chosen loss function, which can be easier to compute than MLE.

Existing literature usually refer to the distribution of $G(Z)$ as the **push-forward** of reference distribution ρ. Such an implicit generative learning approach also enjoys an additional benefit: We only need to choose ρ that is convenient to sample, e.g., uniform or Gaussian distribution, and we then can generate new samples from our learned distribution directly through the estimated mapping G very efficiently.

For many applications, the target distribution can be quite complicated, in contrast to the distribution ρ being simple. This actually requires the mapping G to be flexible. Therefore, we choose to represent G using deep neural networks (DNNs), which are well known for its universal approximation property, i.e., DNNs with sufficiently many neurons and properly chosen activation functions can approximate any continuous functions over compact support up to an arbitrary error. Early empirical evidence, including variational auto-encoder (VAE, [44]) and generative adversarial networks (GAN, [45]) have shown great success of parameterizing G with DNNs. They further motivate a series of variants, which adopt various DNN architectures to learn more complicated generative models [46, 47, 48, 49, 50].

Although the above methods cannot directly estimate the density of the target distribution, for certain applications, we can actually recover the density of $G(Z)$. For example, generative flow methods such as NICE [51], Real NVP [52], and Glow [53]) impose sparsity constraints on weight matrices, and exploit the hierarchical nature of DNNs to compute the densities layer by layer. Specifically, NICE proposed in [51] denotes the transitions of densities within a neural network as

$$Z \xrightarrow{f_0} h_1 \xrightarrow{f_1} h_2 \cdots h_m \xrightarrow{f_m} G(Z),$$

where h_i represents the hidden units of the i-th layer and f_i is the transition function. NICE suggest to restrict the Jacobian matrices of f_i's to be triangular. Therefore, f_i's are reversible and the transition of density in each layer can be easily computed. More recently,

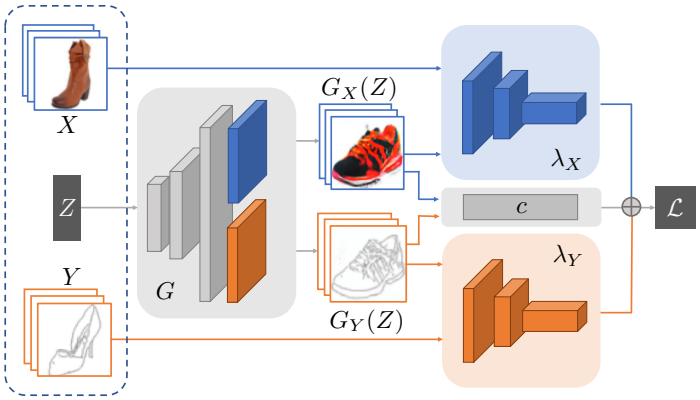


Figure 3.1: An illustration of SPOT.

[40] propose a neural ordinary differential equation (neural ODE) approach to compute the transition from Z to $G(Z)$. Specifically, they introduce a dynamical formulation and parameterizing the mapping G using DNNs with recursive structures: They use an ODE to describe how the input Z gradually evolves towards the output $G(Z)$ in continuous time,

$$dz/dt = \xi(z(t), t),$$

where $z(t)$ denotes the continuous time interpolation of Z, and $\xi(\cdot, \cdot)$ denotes a feedforward-type DNN. Without loss of generality, we choose $z(0) = Z$ and $z(1) = G(Z)$. Then under certain regularity conditions, the mapping $G(\cdot)$ is guaranteed to be reversible, and the density of $G(Z)$ can be computed in $\mathcal{O}(d)$ time, where d is the dimension of Z [54].

3.3 Scalable OT with Pushforward

To achieve better efficiency and scalability, we propose a new framework — named SPOT (Scalable Pushforward of Optimal Transport) — for solving the optimal transport problem. Recall that we aim to find an optimal joint distribution γ given by (Equation 3.1). Let $\mathcal{W}_1(X, \mu)$ denotes the standard Wasserstein metric between a random vector X and a

distribution μ. Specifically, we write

$$\mathcal{W}_1(X, \mu) = \sup_{\lambda_X \in \mathcal{F}^1} \mathbb{E}_X[\lambda_X(X)] - \mathbb{E}_{U\sim\mu}[\lambda_X(U)],$$

where $\mathcal{F}^1$ denotes the class of all 1-Lipschitz functions from $\mathbb{R}^d$ to $\mathbb{R}$. Note that $\mathcal{W}_1(X, \mu) = 0$ indicates $X \sim \mu$. Let $\mathcal{W}_1(Y, \nu)$ be defined analogously as $\mathcal{W}_1(X, \mu)$. Then we can rewrite (Equation 3.1) as

$$\gamma^* = \operatorname*{argmin}_{\gamma} \ \mathbb{E}_{(X,Y)\sim\gamma}[c(X, Y)], \quad \text{subject to} \quad \mathcal{W}_1(X, \mu) = 0, \ \mathcal{W}_1(Y, \nu) = 0. \tag{3.5}$$

As mentioned earlier, solving γ in the space of all continuous distributions is generally intractable. Thus, we adopt the push-forward method, which introduces a mapping G from some latent variable Z to (X, Y). Recall that we denote $(X, Y) = G(Z) = (G_X(Z), G_Y(Z))$ as shown in (Equation 3.2). The latent variable Z follows some distribution ρ that is easy to sample. By the Lagrangian multiplier method and the Kantorovich-Rubinstein duality [55], we then rewrite (Equation 3.5) as

$$\begin{aligned}\min_{G} \max_{\eta_X, \eta_Y, \lambda_X \in \mathcal{F}^1, \lambda_Y \in \mathcal{F}^1} & \mathbb{E}_{Z\sim\rho}[c(G_X(Z), G_Y(Z))] \\ & + \eta_X \mathbb{E}_{Z\sim\rho}[\lambda_X(G_X(Z))] - \mathbb{E}_{U\sim\mu}[\lambda_X(U)] + \eta_Y \mathbb{E}_{Z\sim\rho}[\lambda_Y(G_Y(Z))] - \mathbb{E}_{V\sim\nu}[\lambda_Y(V)].\end{aligned} \tag{3.6}$$

Motivated by [24], we then further parameterize G, λ_X, and λ_Y with neural networks[2]. We denote $\mathcal{G}$ as the class of neural networks for parameterizing G and similarly $\mathcal{F}_X^1$ and $\mathcal{F}_Y^1$ as the classes of 1-Lipschitz functions for λ_X and λ_Y, respectively.

Since $\mathcal{G}$, $\mathcal{F}_X$, and $\mathcal{F}_Y$ are only finite classes, our parameterization of G cannot exactly represent any continuous distributions of (X, Y) (only up to a small approximation error with sufficiently many neurons). Then the marginal distribution constraints, $G_X(Z) \sim \mu$

[2]Using a single neural network to parameterize G encourages parameter sharing between G_X and G_Y. In fact, we can also parameterize G_X and G_Y with different neural networks.

and $G_Y(Z) \sim \nu$, are not necessarily satisfied. Therefore, the Lagrangian multipliers can be unbounded and the equilibrium of (Equation 3.6) does not necessarily exist. To address this issue, we directly treat $\eta_X = \eta_Y = \eta$ as tuning parameters, and solve the following problem instead

$$\min_{G\in\mathcal{G}} \max_{\lambda_X\in\mathcal{F}_X^1,\lambda_Y\in\mathcal{F}_Y^1} \mathbb{E}_{Z\sim\rho}[c(G_X(Z), G_Y(Z))]$$
$$+ \eta\big(\mathbb{E}_{Z\sim\rho}[\lambda_X(G_X(Z))] - \mathbb{E}_{X\sim\mu}[\lambda_X(X)] + \mathbb{E}_{Z\sim\rho}[\lambda_Y(G_Y(Z))] - \mathbb{E}_{Y\sim\nu}[\lambda_Y(Y)]\big). \quad (3.7)$$

We apply alternating stochastic gradient algorithm to solve (Equation 3.7): in each iteration, we perform a few steps of gradient ascent on λ_X and λ_Y, respectively for a fixed G, followed by one-step gradient descent on G for fixed λ_X and λ_Y. We use Spectral Normalization (SN, [56]) to control the Lipschitz constant of λ_X and λ_Y being smaller than 1. Specifically, SN constrains the spectral norm of each weight matrix W by $\text{SN}(W) = W/\sigma(W)$ in every iteration, where $\sigma(W)$ denotes the spectral norm of W. Note that $\sigma(W)$ can be efficiently approximated by a simple one-step power method [57]. Therefore, the computationally intensive SVD can be avoided. We summarize the algorithm in Algorithm Algorithm 3 with SN omitted.

Connection to Wasserstein Generative Adversarial Networks (WGANs): Our proposed framework (Equation 3.7) can be viewed as a multi-task learning version of Wasserstein GANs [39, 58]. Specifically, the mapping G can be viewed as a *generator* that generates samples in the domains $\mathcal{X}$ and $\mathcal{Y}$. The Lagrangian multipliers λ_X and λ_Y can be viewed as *discriminators* that evaluate the discrepancies of the generated sample distributions and the target marginal distributions. By restricting $\lambda_X \in \mathcal{F}_X^1$, $\mathbb{E}_{Z\sim\rho}[\lambda_X(G_X(Z))] - \mathbb{E}_{X\sim\mu}[\lambda_X(X)]$ essentially approximates the Wasserstein distance between the distributions

Algorithm 3 Mini-batch Primal Dual Stochastic Gradient Algorithm for SPOT

Require: Datasets $\{x_i\}_{i=1}^N \sim \mu, \{y_j\}_{j=1}^M \sim \nu$; Initialized networks G, λ_X, and λ_Y with parameters w, θ, and β, respectively; α, the learning rate; n_{critic}, the number of gradient ascent for λ_X and λ_Y; n, the batch size

while w not converged **do**

for $t = 1, 2, \cdots, n_{\text{critic}}$ **do**

Sample mini-batch $\{x_i\}_{i=1}^n$ from $\{x_i\}_{i=1}^N$

Sample mini-batch $\{y_j\}_{j=1}^n$ from $\{y_j\}_{j=1}^M$

Sample mini-batch $\{z_k\}_{k=1}^n$ from ρ

$g_\theta \leftarrow \nabla_\theta(\eta\frac{1}{n}\sum_{k=1}^n \lambda_{X,\theta}(G_{X,w}(z_k)) - \eta\frac{1}{n}\sum_{i=1}^n \lambda_{X,\theta}(x_i))$

$g_\beta \leftarrow \nabla_\beta(\eta\frac{1}{n}\sum_{k=1}^n \lambda_{Y,\beta}(G_{Y,w}(z_k)) - \eta\frac{1}{n}\sum_{i=1}^n \lambda_{Y,\beta}(y_i))$

$\theta \leftarrow \theta + \alpha g_\theta$, $\beta \leftarrow \beta + \alpha g_\beta$

end for

Sample mini-batch $\{z_k\}_{k=1}^n$ from ρ

$g_w \leftarrow \nabla_w(\frac{1}{n}\sum_{k=1}^n c(G_{X,w}(z_k), G_{Y,w}(z_k)) + \eta\frac{1}{n}\sum_{k=1}^n \lambda_{X,\theta}(G_{X,w}(z_k))$
$+\eta\frac{1}{n}\sum_{k=1}^n \lambda_{Y,\beta}(G_{Y,w}(z_k))$

$w \leftarrow w + \alpha g_w$

end while

of $G_X(Z)$ and X under the Euclidean ground cost ([55], the same holds for Y). Denote

$$\mathcal{R}(G_X, G_Y) = \mathbb{E}_{Z\sim\rho}[c(G_X(Z), G_Y(Z))], \quad \text{and}$$

$$d_w(G_X, X) = \max_{\lambda_X \in \mathcal{F}_X^1} \mathbb{E}_{Z\sim\rho}[\lambda_X(G_X(Z))] - \mathbb{E}_{X\sim\mu}[\lambda_X(X)].$$

Let $d_w(G_Y, Y)$ be defined analogously as $d_w(G_X, X)$. We can rewrite (Equation 3.7) as

$$\min_{G\in\mathcal{G}} \eta\big(d_w(G_X, X) + d_w(G_Y, Y)\big) + \mathcal{R}(G_X, G_Y), \tag{3.8}$$

which essentially learns two Wasserstein GANs with a joint generator G through the regularizer $\mathcal{R}$. An illustrative example is provided in Figure Figure 3.1.

Extension to Multiple Marginal Distributions: Our proposed framework can be straightforwardly extended to more than two marginal distributions. Consider the ground cost function c taking m inputs $X_1, \ldots, X_m$ with $X_i \sim \mu_i$ for $i = 1, \ldots, m$. Then the

optimal transport problem (Equation 3.1) becomes the multi-marginal problem [59]:

$$\gamma^* = \underset{\gamma \in \Pi(\mu_1, \mu_2, \cdots, \mu_m)}{\operatorname{argmin}} \mathbb{E}_\gamma[c(X_1, X_2, \cdots, X_m)], \tag{3.9}$$

where $\Pi(\mu_1, \mu_2, \cdots, \mu_m)$ denotes all the joint distributions with marginal distributions satisfying $X_i \sim \mu_i$ for all $i = 1, \ldots, m$. Following the same procedure for two distributions, we cast (Equation 3.9) into the following form

$$\min_{G \in \mathcal{G}} \max_{\lambda_{X_i} \in \mathcal{F}^\eta_{X_i}} \mathbb{E}_{Z\sim\rho}[c(G_{X_1}(Z), \cdots, G_{X_m}(Z))] + \textstyle\sum_{i=1}^m \left(\mathbb{E}_{Z\sim\rho}[\lambda_{X_i}(G_{X_i}(Z))] - \mathbb{E}_{X_i\sim\mu_i}[\lambda_{X_i}(X_i)]\right),$$

where G and λ_{X_i}'s are all parameterized by neural networks. Existing methods for solving the multi-marginal problem (Equation 3.9) suggest to discretize the support of the joint distribution using a refined grid. For complex distributions, the grid size needs to be very large and can be exponential in m [55]. Our parameterization method actually only requires at most $2m$ neural networks, which further corroborates the scalability and efficiency of our framework.

3.4 SPOT for Regularized Density Recovery

Existing literature has shown that entropy-regularized optimal transportation outperforms the un-regularized counterpart in some applications [60, 2]. This is because the entropy regularizer can tradeoff the estimation bias and variance by controlling the smoothness of the density function.

We demonstrate how to efficiently recover the density p_γ of the transport plan with entropy regularization. Instead of parameterizing G by a feedforward neural network, we choose the neural ODE approach, which uses neural networks to approximate the transition from input Z towards output $G(Z)$ in the continuous time. Specifically, we take $z(0) = Z$ and $z(1) = G(Z)$. Let $z(t)$ be the continuous interpolation of Z with density $p(t)$ varying according to time t. We split $z(t)$ into $z_1(t)$ and $z_2(t)$ such that $\dim(z_1) = \dim(X)$ and

$\dim(z_2) = \dim(Y)$. We then write the neural ODE as

$$dz_1/dt = \xi_1(z(t), t), \quad dz_2/dt = \xi_2(z(t), t), \tag{3.10}$$

where ξ_1 and ξ_2 capture the dynamics of $z(t)$. We parameterize $\xi = (\xi_1, \xi_2)$ by a neural network with parameter w. We can describe the dynamics of the joint density $p(t)$ in the following proposition.

Proposition 1. Let z, z_1, z_2, ξ_1 and ξ_2 be defined as above. Suppose ξ_1 and ξ_2 are uniformly Lipschitz continuous in z (the Lipschitz constant is independent of t) and continuous in t. The log joint density satisfies the following ODE:

$$\frac{\partial \log p(t)}{\partial t} = -\left(\operatorname{tr}\left(\frac{\partial \xi_1}{\partial z_1}\right) + \operatorname{tr}\left(\frac{\partial \xi_2}{\partial z_2}\right)\right), \tag{3.11}$$

where $\frac{\partial \xi_1}{\partial z_1}$ and $\frac{\partial \xi_2}{\partial z_2}$ are Jacobian matrices of ξ_1 and ξ_2 with respect to z_1 and z_2, respectively.

Proposition 1 is a direct result of Theorem 1 in [40]. We can now recover the joint density by taking $p_\gamma = p(1)$, which further enables us to efficiently compute the entropy regularizer defined as

$$\mathcal{H}(p_\gamma) = \mathbb{E}_{G(Z)\sim\gamma}[\log p_\gamma(G(Z))].$$

Then we consider the entropy regularized Wasserstein distance $\mathcal{L}_c(G, \lambda_X, \lambda_Y) + \epsilon\mathcal{H}(p_\gamma)$ where $\mathcal{L}_c(G, \lambda_X, \lambda_Y)$ is the objective function in (Equation 3.7). Note that here G is a functional operator of ξ, and hence parameterized with w. The training algorithm follows Algorithm Algorithm 3, except that updating G becomes more complex due to involving the neural ODE and the entropy regularizer.

To update G, we are essentially updating w using the gradient $g_w = \partial(\mathcal{L}_c + \epsilon\mathcal{H})/\partial w$, where ϵ is the regularization coefficient. First we compute $\partial\mathcal{L}_c/\partial w$. We adopt the integral

form from [40] in the following

$$\frac{\partial \mathcal{L}_c}{\partial w} = -\int_0^1 a(t)^\top \frac{\partial \xi(z(t), t)}{\partial w} dt, \tag{3.12}$$

where $a(t) = \partial \mathcal{L}_c / \partial z(t)$ is the so-called "adjoint variable". The detailed derivation is slightly involved due to the complicated terms in the chain rule. We refer the readers to [40] for a complete argument. The advantage of introducing $a(t)$ is that we can compute $a(t)$ using the following ODE,

$$\frac{da(t)}{dt} = -a(t)^\top \frac{\partial \xi(z(t), t)}{\partial z}.$$

Then we can use a well developed numerical method to compute (Equation 3.12) efficiently [61]. Next, we compute $\partial \mathcal{H} / \partial w$ in a similar procedure with $a(t)$ replaced by $b(t) = \partial \mathcal{H} / \partial \log p(t)$. We then write

$$\frac{\partial \mathcal{H}}{\partial w} = -\int_0^1 b(t)^\top \frac{\partial \log p(t)}{\partial w} dt.$$

Using the same numerical method, we can compute $\partial \mathcal{H} / \partial w$, which eventually allows us to compute g_w and update w.

3.5 SPOT for Domain Adaptation

Optimal transport has been used in domain adaptation, but existing methods are either computationally inefficient [62, 63], or cannot achieve a state-of-the-art performance [64]. Here, we demonstrate that SPOT can tackle large scale domain adaptation problems with state-of-the-art performance.

Specifically, we obtain labeled source data $\{x_i\} \sim \mu$, where each data point is associated with a label v_i, and target data $\{y_j\} \sim \nu$ with unknown labels. For simplicity, we use X and Y to denote the random vectors following distributions μ and ν, respectively. The

two distributions μ and ν can be coupled in a way that each paired samples of (X, Y) from the coupled joint distribution are likely to have the same label. In order to identify such coupling information between source and target data, we propose a new OT-based domain adaptation method — DASPOT (Domain Adaptation with SPOT) as follows.

Specifically, we jointly train an optimal transport plan and two classifiers for X and Y (denoted by D_X and D_Y, respectively). Each classifier is a composition of two neural networks — an embedding network and a decision network. For simplicity, we denote $D_X = D_{e,X} \circ D_{c,X}$, where $D_{e,X}$ denotes the embedding network, and $D_{c,X}$ denotes the decision network (respectively for $D_Y = D_{e,Y} \circ D_{c,Y}$). We expect the embedding networks to extract high level features of the source and target data, and then find an optimal transport plan to align X and Y based on these high level features using SPOT. Here we choose a ground cost

$$c(x, y) = \|D_{e,X}(x) - D_{e,Y}(y)\|^2. \tag{3.13}$$

Let G denote the generator of SPOT. The Wasserstein distance of such an OT problem can be written as $\mathbb{E}_Z\|D_{e,X}(G_X(Z)) - D_{e,Y}(G_Y(Z))\|^2$.

Meanwhile, we train D_X by minimizing the empirical risk $\frac{1}{n}\sum_{i=1}^{n}[\mathcal{E}(D_X(x_i), v_i)]$, where $\mathcal{E}$ denotes the cross entropy loss function, and train D_Y by minimizing

$$\mathbb{E}_Z[\mathcal{E}(D_Y(G_Y(Z)), \operatorname*{argmax}_k[D_X(G_X(Z))]_k)], \tag{3.14}$$

where $[v]_k$ denotes the k-th entry of the vector v. The risk function defined in (Equation 3.14) essentially encourages D_X and D_Y to predict each paired (synthetic) samples of $(G_X(Z), G_Y(Z))$ to have the same label.

Eventually, the joint training optimize

$$\min_{D_X, D_Y, G} \max_{\lambda_X, \lambda_Y} \mathcal{L}_c(G, \lambda_X, \lambda_Y) + \frac{\eta_{\mathrm{s}}}{n} \sum_{i=1}^{n} [\mathcal{E}(D_X(x_i), v_i)]$$

$$+ \eta_{\mathrm{da}} \mathbb{E}_Z [\mathcal{E}(D_Y(G_Y(Z)), \operatorname*{argmax}_k [D_X(G_X(Z))]_k)],$$

where $\mathcal{L}_c(G, \lambda_X, \lambda_Y)$ is the objective function in (Equation 3.7) with c defined in (Equation 3.13), and $\eta_{\mathrm{s}}, \eta_{\mathrm{da}}$ are the tuning parameters.

3.6 Experiments

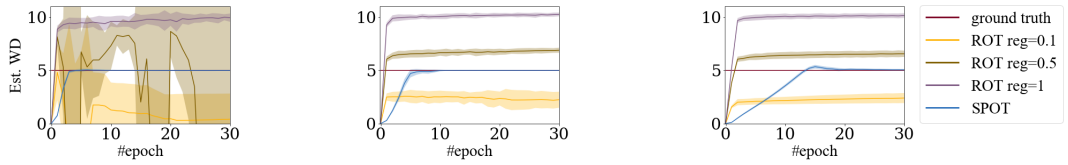
We evaluate the SPOT framework on various tasks: Wasserstein distance approximation, density recovery, paired sample generation and domain adaptation. All experiments are implemented with PyTorch using one GTX1080Ti GPU and a Linux desktop computer with 32GB memory, and we adopt the Adam optimizer with configuration parameters 0.5 and 0.999 [65].

3.6.1 Wasserstein Distance (WD) Approximation

We first demonstrate that SPOT can accurately and efficiently approximate the Wasserstein distance. We take the Euclidean ground cost, i.e. $c(x, y) = \|x - y\|$. Then

$$\mathbb{E}_{G(Z) \sim \gamma^*} [c(G_X(Z), G_Y(Z))]$$

essentially approximates the Wasserstein distance. We take the marginal distributions μ and ν as two Gaussian distributions in $\mathbb{R}^2$ with the same identity covariance matrix. The means are $(-2.5, 0)^\top$ and $(2.5, 0)^\top$, respectively. We find the Wasserstein distance between μ and ν equal to 5 by evaluating its closed-form solution. We generate $n = 10^5$ samples from both distributions μ and ν, respectively. Note that naively applying discretization-based algorithms by dividing the support according to samples requires at least 40 GB memory,



(a) LR =10^{-3} (b) LR =10^{-4} (c) LR =10^{-5}

Figure 3.2: Comparison of convergence between SPOT and ROT. All the curves are averaged over 50 runs with different random seeds, and the shaded areas represent the standard deviation.

which is beyond the memory capability.

We parameterize G_X, G_Y, λ_X, and λ_Y with fully connected neural networks without sharing parameters. All the networks use the Leaky-ReLU activation [66]. G_X and G_Y have 2 hidden layers. λ_X and λ_Y have 1 hidden layer. The latent variable Z follows the standard Gaussian distribution in $\mathbb{R}^2$. We take the batch size equal to 100.

WD vs. Number of Epochs. We compare the algorithmic behavior of SPOT and Regularized Optimal Transport (ROT, [34]) with different regularization coefficients. For SPOT, we set the number of units in each hidden layer equal to 8 and $\eta = 10^4$. For ROT, we adopt the code from the authors[3] with only different input samples, learning rates, and regularization coefficients.

Figure Figure 3.2 shows the convergence behavior of SPOT and ROT for approximating the Wasserstein distance between μ and ν with different learning rates. We observe that SPOT converges to the true Wasserstein distance with only 0.6%, 0.3%, and 0.3% relative errors corresponding to Learning Rates (LR) 10^{-3}, 10^{-4}, and 10^{-5}, respectively. In contrast, ROT is very sensitive to its regularization coefficient. Thus, it requires extensive tuning to achieve a good performance.

WD vs. Number of Hidden Units. We then explore the adaptivity of SPOT by increasing the network size, while the input data are generated from some low dimensional distribution. Specifically, the number of hidden units per layer varies from 2 to 2^{10}. Recall that we parameterize G with two 2-hidden-layer neural networks, and λ_X, λ_Y with two 1-hidden-layer neural networks. Accordingly, the number of parameters in G varies from 36

[3] https://github.com/vivienseguy/Large-Scale-OT

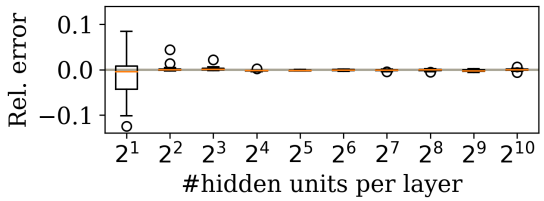


Figure 3.3: Box plots of relative errors of the estimated Wasserstein distance with respect to the number of hidden units per layer. The results are averaged over 50 independent runs.

to about 2×10^6, and that in λ_X or λ_Y varies from 12 to about $2,000$. The tuning parameter η also varies corresponding to the number of hidden units in λ_X, λ_Y. We use $\eta = 10^5$ for $2^1, 2^2$ and 2^3 hidden units per layer, $\eta = 2 \times 10^4$ for $2^4, 2^5$ and 2^6 hidden units per layer, $\eta = 10^4$ for 2^7 and 2^8 hidden units per layer, $\eta = 2 \times 10^3$ for 2^9, and 2^{10} hidden units per layer.

Figure Figure 3.3 shows the estimated WD with respect to the number of hidden units per layer. For large neural networks that have 2^9 or 2^{10} hidden units per layer, i.e., 5.2×10^5 or 2.0×10^6 parameters, the number of parameters is far larger than the number of samples. Therefore, the model is heavily overparameterized. As we can observe in Figure Figure 3.3, the relative error however, does not increase as the number of parameters grows. This suggests that SPOT is quite robust with respect to the network size.

3.6.2 Density Recovery

We demonstrate that SPOT can effectively recover the joint density with entropy regularization. We adopt the neural ODE approach as described in Section section 3.4. Denote $\phi(a, b)$ as the density of the Gaussian distribution $N(a, b)$. We take the marginal distributions μ and ν as (1) Gaussian distributions $\phi(0, 1)$ and $\phi(2, 0.5)$; (2) mixtures of Gaussian $\frac{1}{2}\phi(-1, 0.5) + \frac{1}{2}\phi(1, 0.5)$ and $\frac{1}{2}\phi(-2, 0.5) + \frac{1}{2}\phi(2, 0.5)$. The ground cost is the Euclidean square function, i.e., $c(x, y) = \|x - y\|^2$. We run the training algorithm for 6×10^5 iterations and in each iteration, we generate 500 samples from μ and ν, respectively. We parameterize ξ with a 3-hidden-layer fully-connected neural network with 64 hidden units per layer, and the latent dimension is 2. We take $\eta = 10^6$.

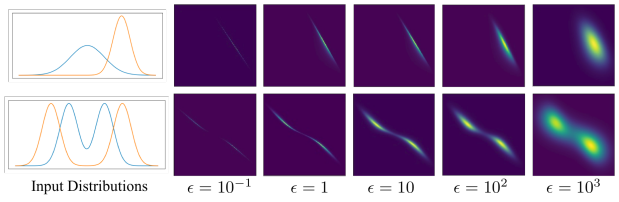


Figure 3.4: Visualization of the marginal distributions and the joint density of the optimal transport plan.

Figure 3.5: Generated samples of SPOT and CoGAN on the MNIST-MNISTM task.

Figure Figure 3.4 shows the input marginal densities and heat maps of output joint densities. We can see that a larger regularization coefficient ϵ yields a smoother joint density for the optimal transport plan. Note that with continuous marginal distributions and the Euclidean square ground cost, the joint density of the unregularized optimal transport degenerates to a generalized impulse function (i.e., a generalized Dirac δ function that has nonzero value on a manifold instead of one atom, as shown in [67, 68]). Entropy regularization prevents such degeneracy by enforcing smoothness of the density.

3.6.3 Sample Generation

We show that SPOT can generate paired samples $(G_X(Z), G_Y(Z))$ from unpaired data X and Y that are sampled from marginal distributions μ and ν, respectively.

Synthetic Data. We take the squared Euclidean cost, i.e. $c(x, y) = \|x - y\|^2$, and adopt the same implementation and sample size as in Section subsection 3.6.1 with learning rate 10^{-3} and 32 hidden units per layer. Figure Figure 3.6 illustrates the input samples and the generated samples with two sets of different marginal distributions: The upper row corresponds to the same Gaussian distributions as in Section subsection 3.6.1. The lower row takes X as Gaussian distribution with mean $(-2.5, 0)^\top$ and covariance $0.5I$, Y as $(\sin(Y_1) + Y_2, 2Y_1 - 3)^\top$, where Y_1 follows a uniform distribution on $[0, 3]$, and Y_2 follows a Gaussian distribution $N(2, 0.1)$.

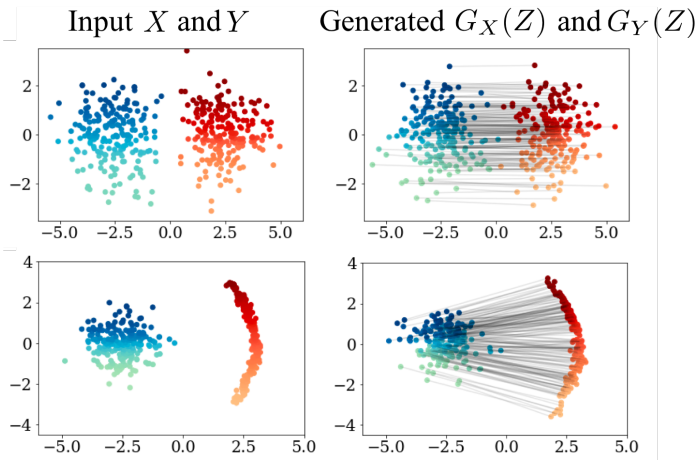


Figure 3.6: Visualization of input samples and generated samples. The black lines represent the paired relation.

We observe that the generated samples and the input samples are approximately identically distributed. Additionally, the paired relationship is as expected – the upper mass is transported to the upper region, and the lower mass is transported to the lower region.

Real Data. We next show SPOT is able to generate high quality paired samples from two unpaired real datasets: MNIST [69] and MNISTM [70]. The handwritten digits in MNIST and MNISTM datasets have different backgrounds and foregrounds (see Figrue Figure 3.5). The digits in paired images however, are expected to have similar contours. We leverage this prior knowledge[4] by adopting a semantic-aware cost function [72] to extract the edge of handwritten letters, i.e., we use the following cost function

$$c(x, y) = \sum_{i=1}^{2} \sum_{j=1}^{3} \| |C_i * x_j| - |C_i * y_j| \|_{\mathrm{F}},$$

where C_1 and C_2 denote the Sobel filter [73], and x_js and y_js are the three channels of RGB

[4]For OT problems, c can be viewed as a way to add prior knowledge to the problem [71].

images. The operator $*$ denotes the matrix convolution. We set

$$C_1 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \text{ and } C_2 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix},$$

with C_1 and C_2 defining two extraction directions.

We now use separate neural networks to parameterize G_X and G_Y instead of taking G_X and G_Y as outputs of a common network. Note that G_X and G_Y does not share parameters. Specifically, we use two 4-layer convolutional layers in each neural network for G_X or G_Y, and two 5-layer convolutional neural networks for λ_X and λ_Y. The batch size is 32, and we train the framework with 2×10^5 iterations until the generated samples become stable.

Figure Figure 3.5 shows the generated samples of SPOT. We also reproduce the results of CoGAN with the code from the authors[5]. As can be seen, with approximately the same network size, SPOT yields paired images with better quality than CoGAN: The contours of the paired results of SPOT are nearly identical, while the results of CoGAN have no clear paired relation. Besides, the images corresponding to $G_Y(Z)$ in SPOT have colorful foreground and background, while in CoGAN there are only few colors. Recall that in SPOT, the paired relation is encouraged by ground cost c, and in CoGAN it is encouraged by sharing parameters. By leveraging prior knowledge in ground cost c, the paired relation is more accurately controlled without compromising the quality of the generated images.

We further test our framework on more complex real datasets: Photo-Monet dataset [74] and Edges-Shoes dataset [75]. We adopt the Euclidean cost function for Photo-Monet dataset, and the semantic-aware cost function as in MNIST-MNISTM for Edges-Shoes dataset. Other implementations remain the same as the MNIST-MINSTM experiment.

Figure Figure 3.7 demonstrates the generated samples of both datasets. We observe that the generated images have a desired paired relation: For each Z, $G_X(Z)$ and $G_Y(Z)$ gives

[5]https://github.com/mingyuliutw/CoGAN

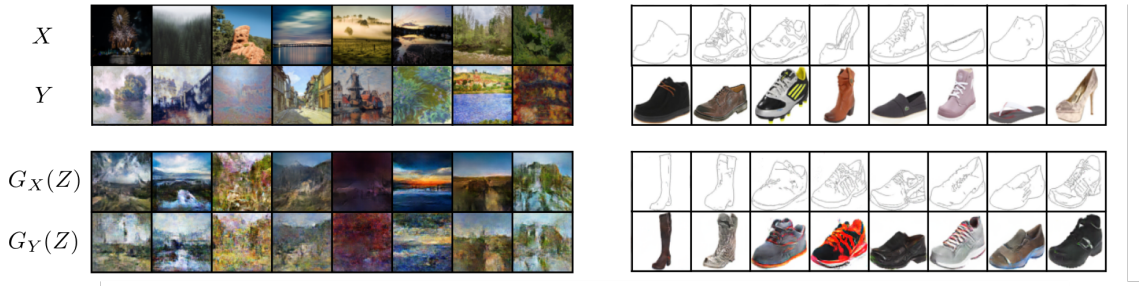


Figure 3.7: Generated samples of SPOT on Photos-Monet and Sketches-Shoes datasets.

Table 3.1: *Domain Adaptation Experiments on multiple tasks.*

Source Target	MNIST USPS	USPS MNIST	SVHN MNIST	MNIST MNISTM
ROT	72.6%	60.5%	62.9%	–
StochJDOT	93.6%	90.5%	67.6%	66.7%
DeepJDOT	95.7%	96.4%	**96.7%**	92.4%
DASPOT	**97.5%**	**96.5%**	96.2%	**94.9%**

a pair of corresponding scenery and shoe. The generated images are also of high quality, especially considering that Photo-Monet dataset is a pretty small but complex dataset with 6,288 photos and 1,073 paintings.

3.6.4 Domain Adaptation

We choose $\eta_{\text{s}} = 10^3$ for all experiments. We set $\eta_{\text{da}} = 0$ for the first 10^5 iteration to wait the generators to be well trained. Then we set $\eta_{\text{da}} = 10$ for the next 3×10^5 iteration. We take totally 4×10^5 iterations, and set the learning rate equal to 10^{-4} and batch size equal to 128 for all experiments.

We evaluate DASPOT with the MNIST, MNISTM, USPS [76], and SVHN [77] datasets. We denote a domain adaptation task as Source Domain $\rightarrow$ Target Domain. For the tasks MNIST $\rightarrow$ USPS, USPS $\rightarrow$ MNIST and MNIST $\rightarrow$ MNISTM, we use three 4-layer networks for D, λ_X,and λ_Y, and two 5-layer networks for G_X and G_Y. For the task SVHN $\rightarrow$ MNIST, we use three 5-layer downsampling ResNets [78] for D, λ_X, and λ_Y, and two 5-layer upsampling ResNets for G_X and G_Y.

We compare the performance of DASPOT with other optimal transport based domain

adaptation methods: ROT [64], StochJDOT [63] and DeepJDOT [63]. As can be seen in Table Table 3.1, DASPOT achieves equal or better performances on all the tasks.

Moreover, we show that DeepJDOT is not as efficient as DASPOT. For example, in the MNIST $\rightarrow$ USPS task, DASPOT requires 169s running time to achieve a 95% accuracy, while DeepJDOT requires 518s running time to achieve the same accuracy. The reason behind is that DeepJDOT needs to solve a series of optimal transport problems using Sinkhorn algorithm. The implementation of DeepJDOT is adapted from the authors' code[6].

3.7 Discussion

Existing literature shows that several stochastic algorithms can efficiently compute the Wasserstein distance between two continuous distributions. These algorithms, however, only apply to the dual of the OT problem (Equation 3.1), and cannot provide the optimal transport plan. For example, [33] suggest to expand the dual variables in two reproducing kernel Hilbert spaces. They then apply the Stochastic Averaged Gradient (SAG) algorithm to compute the optimal objective value of OT with continuous marginal distributions or semi-discrete marginal distributions (i.e., one marginal distribution is continuous and the other is discrete). The follow-up work, [34], parameterize the dual variables with neural networks and apply the Stochastic Gradient Descent (SGD) algorithm to eventually achieve a better convergence. These two methods can only provide the optimal transport plan and recover the joint density when the densities of the marginal distributions are known. This is prohibitive in most applications, since we only have access to the empirical data. Our framework actually allows us to efficiently compute the joint density from the transformation of the latent variable Z as in Section section 3.4.

[6]https://github.com/bbdamodaran/deepJDOT

3.8 Conclusion

We propose the SPOT (Scalable Pushforward of Optimal Transport) framework to efficiently solve optimal transport problems. Specifically, we approximate optimal transport plan as a generative model parameterized by a deep neural network, and cast the optimal transport problem into a minimax optimization problem. We then introduce how to recover the density of the transport plan. Numerical experiments illustrate that SPOT not only has favorable convergence behavior, but is also capable to efficiently generate authentic samples. We finally apply SPOT to domain adaptation problems and achieve state-of-the-art performances.

CHAPTER 4

DIFFERENTIABLE TOP-K OPERATOR WITH OPTIMAL TRANSPORT

4.1 Introduction

The top-k operation, i.e., finding the k largest or smallest elements from a set, is widely used for predictive modeling in information retrieval, machine learning, and data mining. For example, in image retrieval [79, 80, 81], one needs to query the k nearest neighbors of an input image under certain metrics; in the beam search [82, 83] algorithm for neural machine translation, one needs to find the k sequences of largest likelihoods in each decoding step.

Although the ubiquity of top-k operation continues to grow, the operation itself is difficult to be integrated into the training procedure of a predictive model. For example, we consider a neural network-based k-nearest neighbor classifier. Given an input, we use the neural network to extract features from the input. Next, the extracted features are fed into the top-k operation for identifying the k nearest neighbors under some distance metric. We then obtain a prediction based on the k nearest neighbors of the input. In order to train such a model, we choose a proper loss function, and minimize the average loss across training samples using (stochastic) first-order methods. This naturally requires the loss function being differentiable with respect to the input at each update step. Nonetheless, the top-k operation does not exhibit an explicit mathematical formulation: most implementations of the top-k operation, e.g., bubble algorithm and QUICKSELECT [84], involve operations on indices such as indices swapping. Consequently, the training objective is difficult to formulate explicitly.

Alternative perspective — taking the top-k operation as an operator — still cannot

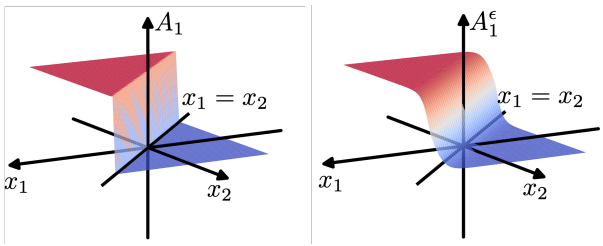


Figure 4.1: Indicator vector with respect to input scores. Left: original top-k operator; right: SOFT top-k operator.

resolve the differentibility issue. Specifically, the top-k operator[1] maps a set of inputs $x_1, \ldots, x_n$ to an index vector $\{0, 1\}^n$. Whereas the Jacobian matrix of such a mapping is not well defined. As a simple example, consider two scalars x_1, x_2. The top-1 operation as in Figure Figure 4.1 returns a vector $[A_1, A_2]^\top$, with each entry denoting whether the scalar is the larger one (1 for true, 0 for false). Denote $A_1 = f(x_1, x_2)$. For a fixed x_2, A_1 jumps from 0 to 1 at $x_1 = x_2$. It is clear that f is not differentiable at $x_1 = x_2$, and the derivative is identically zero otherwise.

Due to the aforementioned difficulty, existing works resort to two-stage training for models with the top-k operation. We consider the neural network-based k-nearest neighbor classifier again. As proposed in [85], one first trains the neural network using some surrogate loss on the extracted features, e.g., using softmax activation in the output layer and the cross-entropy loss. Next, one uses the k-nearest neighbor for prediction based on the features extracted by the well-trained neural network. This training procedure, although circumventing the top-k operation, makes the training and prediction misaligned; and the actual performance suffers.

In this work, we propose the SOFT (Scalable Optimal transport-based diFferenTiable) top-k operation as a differentiable approximation of the standard top-k operation in Section. section 4.2. Specifically, motivated by the implicit differentiation [86, 87, 88, 89] techniques, we first parameterize the top-k operation in terms of the optimal solution of an Optimal Transport (OT) problem. Such a re-parameterization is still not differentiable with respect to the input. To rule out the discontinuity, we impose entropy regularization to the

[1]Throughout the rest of the paper, we refer to the top-k operator as the top-k operation.

optimal transport problem, and show that the optimal solution to the Entropic OT (EOT) problem yields a differentiable approximation to the top-k operation. Moreover, we prove that under mild assumptions, the approximation error can be properly controlled.

We then develop an efficient implementation of the SOFT top-k operation in Section. section 4.3. Specifically, we solve the EOT problem via the Sinkhorn algorithm [2]. Given the optimal solution, we can explicitly formulate the gradient of SOFT top-k operation using the KKT (Karush-Kuhn-Tucker) condition. As a result, the gradient at each update step can be efficiently computed with complexity $\mathcal{O}(n)$, where n is the number of elements in the input set to the top-k operation.

Our proposed SOFT top-k operation allows end-to-end training, and we apply SOFT top-k operation to kNN for classification in Section section 4.4, beam search in Section section 4.5 and learning sparse attention for neural machine translation in Section section 4.6. The experimental results demonstrate significant performance gain over competing methods, as an end-to-end training procedure resolves the misalignment between training and prediction.

Notations. We denote $\|\cdot\|_2$ as the ℓ_2 norm of vectors, $\|\cdot\|_{\mathrm{F}}$ as the Frobenius norm of matrices. Given two matrices $B, D \in \mathbb{R}^{n\times m}$, we denote $\langle B, D\rangle$ as the inner product, i.e., $\langle B, D\rangle = \sum_{i=1,j=1}^{n,m} B_{ij}D_{ij}$. We denote $B \odot D$ as the element-wise multiplication of B and D. We denote $\mathbb{1}(\cdot)$ as the indicator function, i.e., the output of $\mathbb{1}(\cdot)$ is 1 if the input condition is satisfied, and is 0 otherwise. For matrix $B \in \mathbb{R}^{n\times m}$, we denote $B_{i,:}$ as the i-th row of the matrix. The softmax function for matrix B is defined as $\mathrm{softmax}_i(B_{ij}) = e^{B_{ij}} / \sum_{\ell=1}^{n} e^{B_{lj}}$. For a vector $b \in \mathbb{R}^n$, we denote $\mathrm{diag}(b)$ as the matrix where the i-th diagonal entries is b_i.

4.2 SOFT Top-k Operator

In this section we derive the proposed SOFT (Scalable Optimal transport-based diFferenTialble) top-k operator.

4.2.1 Problem Statement

Given a set of scalars $\mathcal{X} = \{x_i\}_{i=1}^n$, the standard top-$k$ operator returns a vector $A = [A_1, \ldots, A_n]^\top$, such that

$$A_i = \begin{cases} 1, & \text{if } x_i \text{ is a top-}k \text{ element in } \mathcal{X}, \\ 0, & \text{otherwise.} \end{cases}$$

In this work, our goal is to design a smooth relaxation of the standard top-k operator, whose Jacobian matrix exists and is smooth. Without loss of generality, we refer to top-k elements as the *smallest* k elements.

4.2.2 Parameterizing Top-k Operator as OT Problem

We first show that the standard top-k operator can be parameterized in terms of the solution of an Optimal Transport (OT) problem [1, 90]. We briefly introduce OT problems for self-containedness. An OT problem finds a transport plan between two distributions, while the expected cost of the transportation is minimized. We consider two discrete distributions defined on supports $\mathcal{A} = \{a_i\}_{i=1}^n$ and $\mathcal{B} = \{b_j\}_{j=1}^m$, respectively. Denote $\mathbb{P}(\{a_i\}) = \mu_i$ and $\mathbb{P}(\{b_j\}) = \nu_j$, and let $\mu = [\mu_1, \ldots, \mu_n]^\top$ and $\nu = [\nu_1, \ldots, \nu_m]^\top$. We further denote $C \in \mathbb{R}^{n \times m}$ as the cost matrix with C_{ij} being the cost of transporting mass from a_i to b_j. An OT problem can be formulated as

$$\Gamma^* = \underset{\Gamma \geq 0}{\operatorname{argmin}} \langle C, \Gamma \rangle, \quad \text{s.t., } \Gamma \mathbf{1}_m = \mu, \ \Gamma^\top \mathbf{1}_n = \nu, \tag{4.1}$$

where $\mathbf{1}$ denotes a vector of ones. The optimal solution Γ^* is referred to as the *optimal transport plan*.

In order to parameterize the top-k operator using the optimal transport plan Γ^*, we set

the support $\mathcal{A} = \mathcal{X}$ and $\mathcal{B} = \{0, 1\}$ in (Equation 4.1), with μ, ν defined as

$$\mu = \mathbf{1}_n/n, \quad \nu = [k/n, (n-k)/n]^\top.$$

We take the cost to be the squared Euclidean distance, i.e., $C_{i1} = x_i^2$ and $C_{i2} = (x_i - 1)^2$ for $i = 1, \ldots, n$. We then establish the relationship between the output A of the top-k operator and Γ^*.

Proposition 2. Consider the setup in the previous paragraph. Without loss of generality, we assume $\mathcal{X}$ has no duplicates. Then the optimal transport plan Γ^* of (Equation 4.1) is

$$\Gamma^*_{\sigma_i,1} = \begin{cases} 1/n, & \text{if } i \le k, \\ 0, & \text{if } k+1 \le i \le n. \end{cases} \tag{4.2}$$

$$\Gamma^*_{\sigma_i,2} = \begin{cases} 0, & \text{if } i \le k, \\ 1/n, & \text{if } k+1 \le i \le n, \end{cases} \tag{4.3}$$

with σ being the sorting permutation, i.e., $x_{\sigma_1} < x_{\sigma_2} < \cdots < x_{\sigma_n}$. Moreover, we have

$$A = n\Gamma^* \cdot [1, 0]^\top. \tag{4.4}$$

Proof. We expand the objective function of (Equation 4.1) as

$$\begin{aligned} \langle C, \Gamma \rangle &= \sum_{i=1}^n \Big((x_i - 0)^2 \Gamma_{i,1} + (x_i - 1)^2 \Gamma_{i,2} \Big) \\ &= \sum_{i=1}^n \Big(x_i^2 (\Gamma_{i,1} + \Gamma_{i,2}) + \Gamma_{i,2} - 2 x_i \Gamma_{i,2} \Big) \\ &= \frac{1}{n} \sum_{i=1}^n x_i^2 + \frac{n-k}{n} - 2 \sum_{i=1}^n x_i \Gamma_{i,2}. \end{aligned}$$

Therefore, to minimize $\langle C, \Gamma \rangle$, it suffices to maximize $\sum_{i=1}^n x_i \Gamma_{i,2}$. It is straightforward to

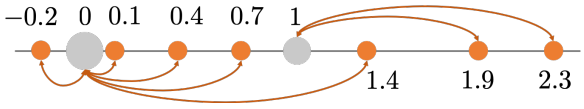


Figure 4.2: Illustration of the optimal transport plan with input $\mathcal{X} = [0.4, 0.7, 2.3, 1.9, -0.2, 1.4, 0.1]^\top$ and $k = 5$. Here, we set $\nu = [\frac{5}{7}, \frac{2}{7}]^\top$. In this way, 5 of the 7 scores, i.e., $\{0.4, 0.7, -0.2, 1.4, 0.1\}$, would align with 0, while $\{2.3, 1.9\}$ align with 1.

check

$$\sum_{i=1}^{n} \Gamma_{i,2} = \frac{n-k}{n} \quad \text{and} \quad \Gamma_{i,2} \leq \frac{1}{n}$$

for any $i = 1, \ldots, n$. Hence, maximizing $\sum_{i=1}^{n} x_i \Gamma_{i,2}$ is essentially selecting the largest $n - K$ elements from $\mathcal{X}$, and the maximum is attained at

$$\Gamma^*_{\sigma_i,2} = \begin{cases} 0, & \text{if } i \leq k, \\ 1/n, & \text{if } k+1 \leq i \leq n. \end{cases}$$

The constraint $\Gamma \mathbf{1}_m = \mu$ further implies that $\Gamma^*_{i,1}$ satisfies (Equation 4.2). Thus, A can be parameterized as $A = n\Gamma^* \cdot [1, 0]^\top$. □

Figure Figure 4.2 illustrates the corresponding optimal transport plan for parameterizing the top-5 operator applied to a set of 7 elements. As can be seen, the mass from the 5 closest points is transported to 0, and meanwhile the mass from the 2 remaining points is transported to 1. Therefore, the optimal transport plan exactly indicates the top-5 elements.

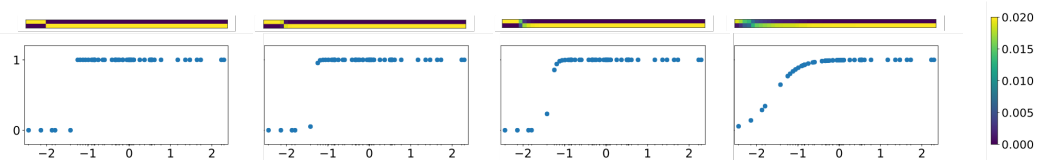
4.2.3 Smoothing by Entropy Regularization

We next rule out the discontinuity of (Equation 4.1) to obtain a smoothed approximation to the standard top-k operator.

Specifically, we employ entropy regularization to the OT problem (Equation 4.1):

$$\Gamma^{*,\epsilon} = \underset{\Gamma \geq 0}{\operatorname{argmin}} \langle C, \Gamma \rangle + \epsilon H(\Gamma), \tag{4.5}$$

$$\text{s.t.,} \quad \Gamma \mathbf{1}_m = \mu, \Gamma^\top \mathbf{1}_n = \nu,$$



(a) $\epsilon = 10^{-3}$ (b) $\epsilon = 5 \times 10^{-3}$ (c) $\epsilon = 10^{-2}$ (d) $\epsilon = 5 \times 10^{-2}$

Figure 4.3: Color maps of Γ^ϵ (upper) and the corresponding scatter plots of values in A^ϵ (lower), where $\mathcal{X}$ contains 50 standard Gaussian samples, and $K = 5$. The scatter plots show the correspondence of the input $\mathcal{X}$ and output A^ϵ.

where $h(\Gamma) = \sum_{i,j} \Gamma_{ij} \log \Gamma_{ij}$ is the entropy regularizer. We define $A^\epsilon = n\Gamma^{*,\epsilon} \cdot [0, 1]^\top$ as a smoothed counterpart of output A in the standard top-k operator. Accordingly, SOFT top-k operator is defined as the mapping from $\mathcal{X}$ to A^ϵ. We show that the Jacobian matrix of SOFT top-k operator exists and is nonzero in the following theorem.

Theorem 3. For any $\epsilon > 0$, SOFT top-k operator: $\mathcal{X} \mapsto A^\epsilon$ is differentiable, as long as the cost C_{ij} is differentiable with respect to x_i for any i, j. Moreover, the Jacobian matrix of SOFT top-k operator always has a nonzero entry for any $\mathcal{X} \in \mathbb{R}^n$.

The proof can be found in Appendix section C.1. We remark that the entropic OT (Equation 4.5) is computationally more friendly, since it allows the usage of first-order algorithms [2].

The Entropic OT introduces bias to the SOFT top-k operator. The following theorem shows that such a bias can be effectively controlled.

Theorem 4. Given a distinct sequence $\mathcal{X}$ and its sorting permutation σ, with Euclidean square cost function, for the proposed top-k solver we have

$$\|\Gamma^{*,\epsilon} - \Gamma^*\|_{\mathrm{F}} \leq \frac{\epsilon(\ln n + \ln 2)}{n(x_{\sigma_{k+1}} - x_{\sigma_k})}.$$

Therefore, with a small enough ϵ, the output vector A^ϵ can well approximate A, especially when there is a large gap between x_{σ_k} and $x_{\sigma_{k+1}}$. Besides, Theorem 5 suggests a trade-off between the bias and regularization of SOFT top-k operator. See Section section 4.8 for a detailed discussion.

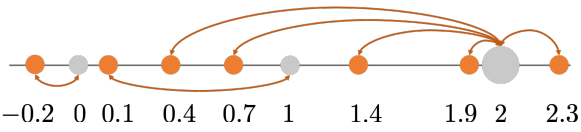


Figure 4.4: Illustration of the optimal transport plan for sorted top-k with input $\mathcal{X} = [0.4, 0.7, 2.3, 1.9, -0.2, 1.4, 0.1]^\top$ and $K = 2$. Here, we set $\nu = [\frac{1}{7}, \frac{1}{7}, \frac{5}{7}]^\top$ and $\mathcal{B} = [0, 1, 2]^\top$. In this way, the smallest score -0.2 aligns with 0, the second smallest score 0.1 aligns with 1, and the rest of the scores align with 2.

4.2.4 Sorted SOFT Top-k Operator

In some applications like beam search, we not only need to distinguish the top-k elements, but also sort the top-k elements. For example, in image retrieval [81], the retrieved k images are expected to be sorted. We show that our proposed SOFT top-k operator can be extended to the sorted SOFT top-k operator.

Analogous to the derivation of the SOFT top-k operator, we first parameterize the sorted top-k operator in terms of an OT problem. Specifically, we keep $\mathcal{A} = \mathcal{X}$ and $\mu = \mathbf{1}_n/n$ and set

$$\mathcal{B} = [0, 1, 2, \cdots, k]^\top, \text{ and}$$
$$\nu = [1/n, \cdots, 1/n, (n-k)/n]^\top.$$

One can check that the optimal transport plan of the above OT problem transports the smallest element in $\mathcal{A}$ to 0 in $\mathcal{B}$, the second smallest element to 1, and so on so forth. This in turn yields the sorted top-k elements. Figure Figure 4.4 illustrates the sorted top-2 operator and its corresponding optimal transport plan.

The sorted SOFT top-k operator is obtained similarly to SOFT top-k operator by solving the entropy regularized OT problem. We can show that the sorted SOFT top-k operator is differentiable and the bias can be properly controlled.

4.3 Efficient Implementation

We now present our implementation of SOFT top-k operator, which consists of 1) computing A^ϵ from $\mathcal{X}$ and 2) computing the Jacobian matrix of A^ϵ with respect to $\mathcal{X}$. We refer to 1) as the forward pass and 2) as the backward pass.

Forward Pass. The forward pass from $\mathcal{X}$ to A^ϵ can be efficiently computed using Sinkhorn algorithm. Specifically, we run iterative Bregman projections [3], where at the ℓ-th iteration, we update

$$p^{(\ell+1)} = \frac{\mu}{Gq^{(\ell)}}, \quad q^{(\ell+1)} = \frac{\nu}{G^\top p^{(\ell+1)}}.$$

Here the division is entrywise, $q^{(0)} = \mathbf{1}_2/2$, and $G \in \mathbb{R}^{n\times m}$ with $G_{ij} = e^{-\frac{C_{ij}}{\epsilon}}$. Denote p^* and q^* as the stationary point of the Bregman projections. The optimal transport plan $\Gamma^{*,\epsilon}$ can be otained by $\Gamma^{*,\epsilon}_{ij} = p^*_i G_{ij} q^*_j$. The algorithm is summarized in Algorithm Algorithm 4.

Algorithm 4 SOFT Top-k

Require: $\mathcal{X} = [x_i]_{i=1}^n, k, \epsilon, L$
$\mathcal{Y} = [y_1, y_2]^\top = [0, 1]^\top$
$\mu = \mathbf{1}_n/n, \nu = [k/n, (n-K)/n]^\top$
$C_{ij} = |x_i - y_j|^2, G_{ij} = e^{-\frac{C_{ij}}{\epsilon}}, q = \mathbf{1}_2/2$
for $l = 1, \cdots, L$ **do**
$p = \mu/(Gq), q = \nu/(G^\top p)$
end for
$\Gamma = \mathrm{diag}(p) \odot G \odot \mathrm{diag}(q)$
$A^\epsilon = n\Gamma \cdot [0, 1]^\top$

Backward Pass. Given A^ϵ, we compute the Jacobian matrix $\frac{dA^\epsilon}{d\mathcal{X}}$ using implicit differentiation and differentiable programming techinques. Specifically, the Lagrangian function of Problem (Equation 4.5) is

$$\mathcal{L} = \langle C, \Gamma\rangle - \xi^\top(\Gamma\mathbf{1}_m - \mu) - \zeta^\top(\Gamma^\top\mathbf{1}_n - \nu) + \epsilon H(\Gamma),$$

where ξ and ζ are dual variables. The KKT condition implies that the optimal solution $\Gamma^{*,\epsilon}$ can be formulated using the optimal dual variables ξ^* and ζ^* as (Sinkhorn's scaling theorem, [91]),

$$\Gamma^{*,\epsilon} = \mathrm{diag}(e^{\frac{\xi^*}{\epsilon}})e^{-\frac{C}{\epsilon}}\mathrm{diag}(e^{\frac{\zeta^*}{\epsilon}}). \tag{4.6}$$

Substituting (Equation 4.6) into the Lagrangian function, we obtain

$$\mathcal{L}(\xi^*, \zeta^*; C) = (\xi^*)^\top \mu + (\zeta^*)^\top \nu - \epsilon \sum_{i,j=1}^{n,m} e^{-\frac{C_{ij}-\xi_i^*-\zeta_j^*}{\epsilon}}.$$

We now compute the gradient of ξ^* and ζ^* with respect to C, such that we can obtain $d\Gamma^{*,\epsilon}/dC$ by the chain rule applied to (Equation 4.6). Denote $\omega^* = [(\xi^*)^\top, (\zeta^*)^\top]^\top$, and $\phi(\omega^*; C) = \partial\mathcal{L}(\omega^*; C)/\partial\omega^*$. At the optimal dual variable ω^*, the KKT condition immediately yields

$$\phi(\omega^*; C) \equiv 0.$$

By the chain rule, we have

$$\frac{d\phi(\omega^*; C)}{dC} = \frac{\partial\phi(\omega^*; C)}{\partial C} + \frac{\partial\phi(\omega^*; C)}{\partial\omega^*}\frac{d\omega^*}{dC} = 0.$$

Rerranging terms, we obtain

$$\frac{d\omega^*}{dC} = -\left(\frac{\partial\phi(\omega^*; C)}{\partial\omega^*}\right)^{-1}\frac{\partial\phi(\omega^*; C)}{\partial C}. \tag{4.7}$$

Combining (Equation 4.6), (Equation 4.7), $C_{ij} = (x_i - y_j)^2$, and $A^\epsilon = n\Gamma^{*,\epsilon} \cdot [1, 0]^\top$, the Jacobian matrix $dA^\epsilon/d\mathcal{X}$ can then be derived using the chain rule again.

The detailed derivation and the corresponding algorithm for computing the Jacobian matrix can be found in Appendix section C.2. The time and space complexity of the derived

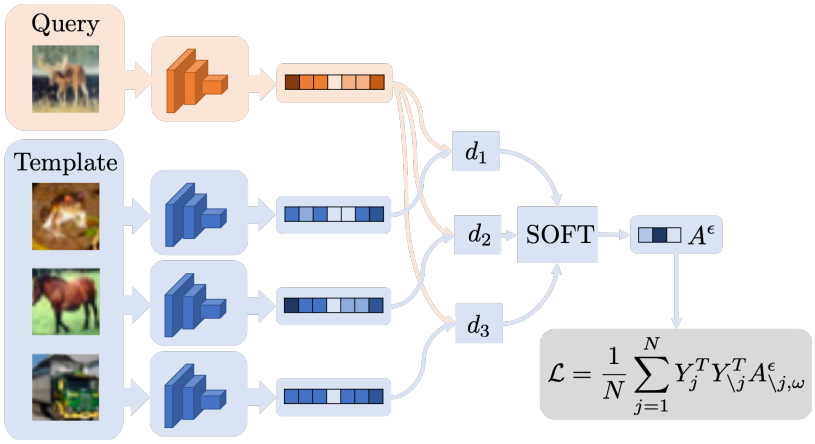


Figure 4.5: Illustration of the entire forward pass of kNN.

algorithm is $\mathcal{O}(n)$ and $\mathcal{O}(kn)$ for top-k and sorted top-k operators, respectively. We also include a Pytorch [92] implementation of the forward and backward pass in Appendix section C.2 by extending the `autograd` automatic differentiation package.

4.4 k-NN for Image Classification

The proposed SOFT top-k operator enables us to train an end-to-end neural network-based kNN classifier. Specifically, we receive training samples $\{Z_i, y_i\}_{i=1}^N$ with Z_i being the input data and $y_i \in \{1, \ldots, M\}$ the label from M classes. During the training, for an input data Z_j (also known as the query sample), we associate a loss as follows. Denote $Z_{\setminus j}$ as all the input data excluding Z_j (also known as the template samples). We use a neural network to extract features from all the input data, and measure the pairwise Euclidean distances between the extracted features of $Z_{\setminus j}$ and that of Z_j. Denote $\mathcal{X}_{\setminus j,\theta}$ as the collection of these pairwise distances, i.e.,

$$\mathcal{X}_{\setminus j,\theta} = \{\|f_\theta(Z_1) - f_\theta(Z_j)\|_2, ..., \|f_\theta(Z_{j-1}) - f_\theta(Z_j)\|_2,$$
$$\|f_\theta(Z_{j+1}) - f_\theta(Z_j)\|_2, ..., \|f_\theta(Z_N) - f_\theta(Z_j)\|_2\},$$

where f_θ is the neural network parameterized by θ, and the subscript of $\mathcal{X}$ emphasizes its dependence on θ.

Next, we apply SOFT top-k operator to $\mathcal{X}_{\setminus j,\omega}$, and the returned vector is denoted by $A^{\epsilon}_{\setminus j,\theta}$. Let $Y_{\setminus j} \in \mathbb{R}^{M\times(N-1)}$ be the matrix by concatenating the one-hot encoding of labels y_i for $i \neq j$ as columns, and $Y_j \in \mathbb{R}^M$ the one-hot encoding of the label y_j. The loss of Z_j is defined as

$$\ell(Z_j, y_j) = Y_j^\top Y_{\setminus j}^\top A^{\epsilon}_{\setminus j,\theta}.$$

Consequently, the training loss is

$$\mathcal{L}(\{Z_j, y_j\}_{j=1}^N) = \frac{1}{N}\sum_{j=1}^N \ell(Z_j, y_j) = \frac{1}{N}\sum_{j=1}^N Y_j^\top Y_{\setminus j}^\top A^{\epsilon}_{\setminus j,\theta}.$$

Recall that the Jacobian matrix of $A^{\epsilon}_{\setminus j,\theta}$ exists and has no zero entries. This allows us to utilize stochastic gradient descent algorithms to update θ in the neural network. Moreover, since N is often large, to ease the computation, we randomly sample a batch of samples to compute the stochastic gradient at each iteration.

In the prediction stage, we use all the training samples to obtain a predicted label of a query sample. Specifically, we feed the query sample into the neural network to extract its features, and compute pairwise Euclidean distances to all the training samples. We then run the standard kNN algorithm [93] to obtain the predicted label.

4.4.1 Experiment

We evaluate the performance of the proposed neural network-based kNN classifier on two benchmark datasets: MNIST dataset of handwritten digits [69] and the CIFAR-10 dataset of natural images [95] with the canonical splits for training and testing without data augmentation. We adopt the coefficient of entropy regularizer $\epsilon = 10^{-3}$ for MNIST dataset and $\epsilon = 10^{-5}$ for CIFAR-10 dataset. Detailed settings of the model and training procedure are deferred to Appendix section C.3.

Baselines. We consider several baselines:

Table 4.1: Classification accuracy of kNN.

Algorithm	MNIST	CIFAR10
kNN	97.2%	35.4%
kNN+PCA	97.6%	40.9%
kNN+AE	97.6%	44.2%
kNN+pretrained CNN	98.4%	91.1%
RelaxSubSample	99.3%	90.1%
kNN+NeuralSort	$\mathbf{99.5}\%$	90.7%
kNN+[94]	99.0%	84.8%
kNN+Softmax k times	99.3%	92.2%
CE+CNN [78]	99.0%	91.3%
kNN+**SOFT Top-k**	99.4%	$\mathbf{92.6}\%$

1. Standard kNN method.

2. Two-stage training methods: we first extract the features of the images, and then perform kNN on the features. The feature is extracted using Principle Component Analysis (PCA, top-50 principle components is adopted), autoencoder (AE), or a pretrained Convolutional Neural Network (CNN) using the Cross-Entropy (CE) loss.

3. Differentiable *ranking* + kNN: This includes NeuralSort [96] and [94]. [94] is not directly applicable, which requires some adaptation (see Appendix section C.3).

4. Stochastic kNN with Gumbel top-k relaxation [97]: The model is referred as Relax-SubSample.

5. Softmax Augmentation for smoothed top-k operation: A combination of k softmax operation is used to replace the top-k operator. Specifically, we recursively perform softmax on $\mathcal{X}$ for k times (Similar idea appears in [98]). At the k-th iteration, we mask the top-$(k-1)$ entries with negative infinity.

6. CNNs trained with CE without any top-k component[2].

For the pretrained CNN and CNN trained with CE, we adopt identical neural networks as our method.

[2]Our implementation is based on github.com/pytorch/vision.git

Results. We report the classification accuracies on the standard test sets in Table Table 4.1. On both datasets, the SOFT kNN classifier achieves comparable or better accuracies.

4.5 Beam Search for Machine Translation

Beam search is a popular method for the *inference* of Neural Language Generation (NLG) models, e.g., machine translation models. Here, we propose to incorporate beam search into the *training* procedure based on SOFT top-k operator.

4.5.1 Misalignment between Training and Inference

Denote the predicted sequence as $y = [y^{(1)}, \cdots, y^{(T)}]$, and the vocabularies as $\{z_1, \cdots, z_V\}$. Consider a recurrent network based NLG model. The output of the model at the t-th decoding step is a probability simplex $[\mathbb{P}(y^{(t)} = z_i|h^{(t)}]_{i=1}^V$, where $h^{(t)}$ is the hidden state associated with the sequence $y^{(1:t)} = [y^{(1)}, ..., y^{(t)}]$.

Beam search recursively keeps the sequences with the k largest likelihoods, and discards the rest. Specifically, at the $(t+1)$-th decoding step, we have k sequences $\widetilde{y}^{(1:t),i}$'s obtained at the t-th step, where $i = 1, ..., k$ indexes the sequences. The likelihood of $\widetilde{y}^{(1:t),i}$ is denoted by $\mathcal{L}_{\mathrm{s}}(\widetilde{y}^{(1:t),i})$. We then select the next k sequences by varying $i = 1, \ldots, k$ and $j = 1, \ldots, V$:

$$\{\widetilde{y}^{(1:t+1),\ell}\}_{\ell=1}^{k} = \arg\,\text{top-k}_{[\widetilde{y}^{(1:t),i},z_j]}\mathcal{L}_{\mathrm{s}}([\widetilde{y}^{(1:t),i}, z_j]).$$

where $\mathcal{L}_{\mathrm{s}}([\widetilde{y}^{(1:t),i}, z_j])$ is the likelihood of the sequence appending z_j to $\widetilde{y}^{(1:t),i}$ defined as

$$\mathcal{L}_{\mathrm{s}}([\widetilde{y}^{(1:t),i}, z_j]) = \mathbb{P}(y^{(t+1)} = z_j|h^{(t+1),i})\mathcal{L}_{\mathrm{s}}(\widetilde{y}^{(1:t),i}), \tag{4.8}$$

and $h^{(t+1),i}$ is the hidden state generated from $\widetilde{y}^{(1:t),i}$. Note that z_j's and $\widetilde{y}^{(1:t),i}$'s together yield Vk choices. Here we abuse the notation: $\widetilde{y}^{(1:t+1),\ell}$ denotes the ℓ-th selected sequence

at the $(t+1)$-th decoding step, and is not necessarily related to $\widetilde{y}^{(1:t),i}$ at the t-th decoding step, even if $i = \ell$.

For $t = 1$, we set $\widetilde{y}^{(1)} = z_{\mathrm{s}}$ as the start token, $\mathcal{L}_{\mathrm{s}}(y^{(1)}) = 1$, and $h^{(1)} = h_{\mathrm{e}}$ as the output of the encoder. We repeat the above procedure, until the end token is selected or the pre-specified max length is reached. At last, we select the sequence $y^{(1:T),*}$ with the largest likelihood as the predicted sequence.

Moreover, the most popular training procedure for NLG models directly uses the so-called "*teacher forcing*" framework. As the ground truth of the target sequence (i.e., gold sequence) $\bar{y} = [\bar{y}^{(1)}, \cdots, \bar{y}^{(T)}]$ is provided at the training stage, we can directly maximize the likelihood

$$\mathcal{L}_{\mathrm{tf}} = \prod_{t=1}^{T} \mathbb{P}(y^{(t)} = \bar{y}^{(t)} | h^{(t)}(\bar{y}^{(1:t\text{-}1)})). \tag{4.9}$$

As can be seen, such a training framework only involve the gold sequence, and cannot take the uncertainty of the recursive exploration of the beam search into consideration. Therefore, it yields a misalignment between model training and inference [99], which is also referred as *exposure bias* [83].

4.5.2 Differential Beam Search with Sorted SOFT Top-k

To mitigate the aforementioned misalignment, we propose to integrate beam search into the training procedure, where the top-k operator in the beam search algorithm is replaced with our proposed sorted SOFT top-k operator proposed in Section subsection 4.2.4.

Specifically, at the $(t+1)$-th decoding step, we have k sequences denoted by $E^{(1:t),i}$, where $i = 1, ..., k$ indexes the sequences. Here $E^{(1:t),i}$ consists of a sequence of D-dimensional vectors, where D is the embedding dimension. We are not using the tokens, and the reason behind will be explained later. Let $\widetilde{h}^{(t),i}$ denote the hidden state generated

from $E^{(1:t),i}$. We then consider

$$\mathcal{X}^{(t)} = \{-\mathcal{L}_{\mathrm{s}}([E^{(1:t),i}, w_j]), j = 1, ..., V,\ i = 1, ..., k\},$$

where $\mathcal{L}_{\mathrm{s}}(\cdot)$ is defined analogously to (Equation 4.8), and $w_j \in \mathbb{R}^D$ is the embedding of token z_j.

Recall that ϵ is the smoothing parameter. We then apply the sorted SOFT top-k operator to $\mathcal{X}^{(t)}$ to obtain $\{E^{(1:t+1),\ell}\}_{\ell=1}^{k}$, which are k sequences with the largest likelihoods. More precisely, the sorted SOFT top-k operator yields an output tensor $A^{(t),\epsilon} \in \mathbb{R}^{V\times k\times k}$, where $A^{(t),\epsilon}_{ji,\ell}$ denotes the smoothed indicator of whether $[E^{(1:t),i}, w_j]$ has a rank ℓ. We then obtain

$$E^{(1:t+1),\ell} = \Big[E^{(1:t),r}, \sum_{j=1}^{V}\sum_{i=1}^{k} A^{(t),\epsilon}_{ji,\ell} w_j\Big], \tag{4.10}$$

where r denotes the index i (for $E^{(1:t),i}$'s) associated with the index ℓ (for $E^{(1:t+1),\ell}$'s).

Accordingly, we generate the k hidden states for the $(t+1)$-th decoding step:

$$\widetilde{h}^{(t),\ell} = \sum_{j=1}^{V}\sum_{i=1}^{k} A^{(t),\epsilon}_{ji,\ell} h^{(t),i}, \tag{4.11}$$

where $h^{(t),i}$ is the intermediate hidden state generated by the decoder based on $E^{(1:t),i}$.

After decoding, we select the sequence with largest likelihood $E^{(1:T),*}$, and maximize the likelihood as follows,

$$\mathcal{L}_{\mathrm{SOFT}} = \prod_{t=1}^{T} \mathbb{P}(y^{(t)} = \bar{y}^{(t)} | \widetilde{h}^{(t\text{-}1),*}(E^{(1:t\text{-}1),*})).$$

We provide the sketch of training procedure in Algorithm Algorithm 5, where we denote $\mathrm{logit}^{(t),i}$ as $[\log \mathbb{P}(y^{(t+1)} = \omega_j | \widetilde{h}^{(t),i}(E^{(1:t),i}))]_{j=1}^{V}$, which is part of the output of the decoder. More technical details (e.g., backtracking algorithm for finding the index r in (Equation 4.10)) are provided in Appendix section C.3.

Algorithm 5 Beam search training with SOFT Top-k

Require: Input sequence s, target sequence $\bar{y}$; embedding matrix $W \in \mathbb{R}^{V \times D}$; max length T; k; regularization coefficient ϵ; number of Sinkhorn iteration L
$\widetilde{h}_i^{(1)} = h_{\mathrm{e}} = \text{Encoder}(s)$, $E^{(1),i} = w_{\mathrm{s}}$
for $t = 1, \cdots, T-1$ **do**
  for $i = 1, \cdots, k$ **do**
   $\text{logit}^{(t),i}, h^{(t),i} = \text{Decoder}(E^{(t),i}, \widetilde{h}^{(t),i})$
   $\log \mathcal{L}_{\mathrm{s}}([E^{(1:t),i}, w_j]) = \log \mathcal{L}_{\mathrm{s}}(E^{(1:t),i}) + \text{logit}_j^{(t),i}$
   $\mathcal{X}^{(t)} = \{-\log \mathcal{L}_{\mathrm{s}}([E^{(1:t),i}, w_j]) \mid j = 1, \cdots, V\}$
  end for
  $A^{(t),\epsilon}$ = Sorted-SOFT-Top-k($\mathcal{X}^{(t)}, k, \epsilon, L$)
  Compute $E^{(t+1),\ell}$, $\widetilde{h}^{(t+1),\ell}$ as in (Equation 4.10) and (Equation 4.11)
end for
Compute $\nabla \mathcal{L}_{\mathrm{SOFT}}$ and update the model

Note that integrating the beam search into training essentially yields a very large search space for the model, which is not necessarily affordable sometimes. To alleviate this issue, we further propose a hybrid approach by combining the teacher forcing training with beam search-type training. Specifically, we maximize the weighted likelihood defined as follows,

$$\mathcal{L}_{\mathrm{final}} = \rho \mathcal{L}_{\mathrm{tf}} + (1 - \rho) \mathcal{L}_{\mathrm{SOFT}},$$

where $\rho \in (0, 1)$ is referred to as the "teaching forcing ratio". The teaching forcing loss $\mathcal{L}_{\mathrm{tf}}$ can help reduce the search space and improve the overall performance.

4.5.3 Experiment

We evaluate our proposed beam search + sorted SOFT top-k training procedure using WMT2014 English-French dataset.

Setting. We adopt beam size 5, teacher forcing ratio $\rho = 0.8$, and $\epsilon = 10^{-1}$. For detailed settings of the training procedure, please refer to Appendix section C.3.

We reproduce the experiment in [100], and run our proposed training procedure with the identical data pre-processing procedure and the LSTM-based sequence-to-sequence model. Different from [100], here we also preprocess the data with *byte pair encoding* [101].

Results. As shown in Table Table 4.2, the proposed SOFT beam search training procedure achieves an improvement in BLEU score of approximately 0.9. We also include other LSTM-based models for baseline comparison.

Table 4.2: BLEU scores on WMT'14 with single LSTM model.

Algorithm	BLEU
[102]	33.10
[103]	30.82
[104]	34.54
[105]	30.59
[100]	28.45
[106]	34.60
[100] (Our implementation)	35.38
Beam Search + Sorted SOFT Top-k	**36.27**

4.6 Top-k Attention for Machine Translation

We apply SOFT top-k operator to yield sparse attention scores. Attention module is an integral part of various natural language processing tasks, allowing modeling of long-term and local dependencies. Specifically, given the vector representations of a source sequence $s = [s_1, \cdots, s_N]^\top$ and target sequence $y = [y_1, \cdots, y_M]^\top$, we compute the alignment score between s_i and y_j by a compatibility function $f(s_i, y_j)$, e.g., $f(s_i, y_j) = s_i^\top y_j$, which measures the dependency between s_i and y_j. A softmax function then transforms the scores $[f(s_i, y_j)]_{i=1}^N$ to a sum-to-one weight vector w_j for each y_j. The output o_j of this attention module is a weighted sum of s_i's, i.e., $o_j = w_j^\top s$.

The attention module described above is called the soft attention, i.e., the attention scores w_j of y_j is not sparse. This may lead to redundancy of the attention [107, 108]. Empirical results show that hard attention, i.e., enforcing sparsity structures in the score w_j's, yields more appealing performance [109]. Therefore, we propose to replace the softmax operation on $[f(s_i, y_j)]_{i=1}^N$ by the standard top-k operator to select the top-k elements. In order for an end-to-end training, we further deploy SOFT top-k operator to substitute the

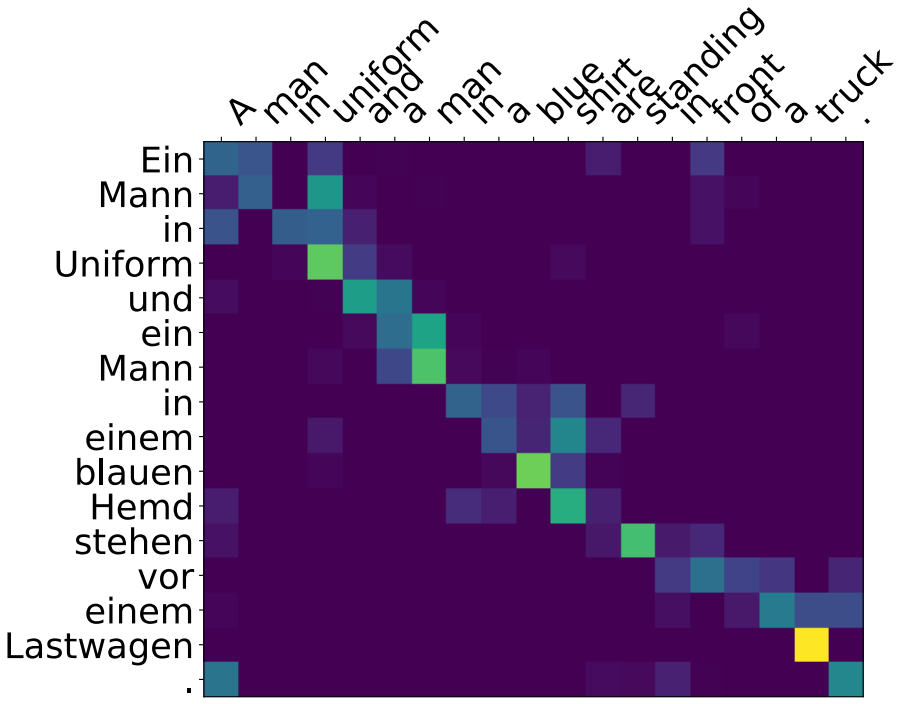


Figure 4.6: Visualization of the top-K attention.

standard top-k operator. Given $[f(s_i, y_j)]_{i=1}^N$, the output of SOFT top-k operator is denoted by A_j^ϵ, and the weight vector w_j is now computed as

$$w_j = \text{softmax}([f(s_1, y_j), \ldots, f(s_N, y_j)]^\top + \log A_j^\epsilon).$$

Here $\log$ is the entrywise logarithm. The output o_j of the attention module is computed the same $o_j = w_j^\top s$. Such a SOFT top-k attention will promote the top-k elements in $[f(s_i, y_j)]_{i=1}^N$ to be even larger than the non-top-k elements, and eventually promote the attention of y_j to focus on k tokens in s.

4.6.1 Experiment

We evaluate the proposed top-k attention on WMT2016 English-German dataset. Our implementation and settings are based on [110][3]. For a fair comparison, we implement a standard soft attention using the same settings as the baseline. The details are provided in

[3]Settings on data pre-processing, model, and training procedure is identical to https://opennmt.net/OpenNMT-py/extended.html.

Appendix section C.3.

Results. As shown in Table Table 4.3, the proposed SOFT top-k attention training procedure achieves an improvement in BLEU score of approximately 0.8. We visualize the top-k attention in Figure Figure 4.6. The attention matrix is sparse, and has a clear semantic meaning – "truck" corresponds to "Lastwagen", "blue" corresponds to "blauen", "standing" corresponds to "stehen", etc.

Table 4.3: BLEU scores on WMT'16.

Algorithm	BLEU
Proposed Top-k Attention	$\mathbf{37.30}$
Soft Attention	36.54

4.7 Related Work

We parameterize the top-k operator as an optimal transport problem, which shares the same spirit as [94]. Specifically, [94] formulate the *ranking* problem as an optimal transport problem. Ranking is more complicated than identifying the top-k elements, since one needs to align different ranks to corresponding elements. Therefore, the algorithm complexity per iteration is $\mathcal{O}(n^2)$ in [94] (see first experiment in their section 4), while we attain $\mathcal{O}(n)$ complexity for the easier top-k operator. Besides methodology, we explicitly characterize the bias of SOFT top-k operator to the standard counterpart, and our efficient implementation allows us to perform more complex experiments such as beam search.

Gumbel-Softmax trick [111] can also be utilized to derive a continuous relaxation of the top-k operator. Specifically, [112] adapted such a trick to sample k elements from n choices, and [97] further applied the trick to stochastic kNN, where neural networks are used to approximating the sorting operator. However, as shown in our experiments (see Table Table 4.1), the performance of stochastic kNN is not as good as deterministic kNN.

4.8 Discussion

Relation to automatic differentiation. We compute the Jacobian matrix of SOFT top-k operator with respect to its input using the optimal transport plan of the entropic OT problem (Equation 4.5) in the backward pass. The optimal transport plan can be obtained by the Sinkhorn algorithm (Algorithm Algorithm 4), which is iterative and each iteration only involves multiplication and addition. Therefore, we can also apply automatic differentiation (auto-diff) to compute the Jacobian matrix. Specifically, we denote Γ_ℓ as the transport plan at the t-th iteration of Sinkhorn algorithm. The update of Γ_ℓ can be written as $\Gamma_{\ell+1} = \mathcal{T}(\Gamma_\ell)$, where $\mathcal{T}$ denotes the update of the Sinkhorn algorithm. In order to apply auto-diff, we need to store all the intermediate states, e.g., p, q, G in each iteration, as defined in Algorithm Algorithm 4 at each iteration. This requires a huge memory size proportional to the total number of iterations of the algorithm. In contrast, our backward pass allows us to save memory.

Bias and regularization trade-off. Theorem 5 suggests a trade-off between the regularization and bias of SOFT top-k operator. Specifically, a large ϵ has a strong smoothing effect on the entropic OT problem, and the corresponding entries of the Jacobian matrix are neither too large nor too small. This eases the end-to-end training process. However, the bias of SOFT top-k operator is large, which can deteriorate the model performance. On the contrary, a smaller ϵ ensures a smaller bias. Yet the SOFT top-k operator is less smooth, which in turn makes the end-to-end training less efficient.

On the other hand, the bias of SOFT top-k operator also depends on the gap between $x_{\sigma_{k+1}}$ and x_{σ_k}. In fact, such a gap can be viewed as the signal strength of the problem. A large gap implies that the top-k elements are clearly distinguished from the rest of the elements. Therefore, the bias is expected to be small since the problem is relatively easy. Moreover, in real applications such as neural network-based kNN classification, the end-to-end training process promotes neural networks to extract features that exhibit a large gap

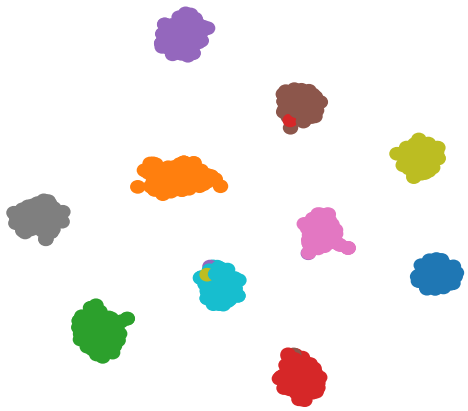


Figure 4.7: Visualization of the MNIST data based on features extracted by the neural network-based k-NN classifier trained by our proposed method in Section section 4.4.

(as illustrated in Figure Figure 4.7). Hence, the bias of SOFT top-k operator can be well controlled in practice.

CHAPTER 5
A HYPERGRADIENT APPROACH TO ROBUST REGRESSION WITHOUT CORRESPONDENCE

5.1 Introduction

Regression analysis has been widely used in various machine learning applications to infer the the relationship between an explanatory random variable (i.e., the input) $X \in \mathbb{R}^d$ and a response random variable (i.e., the output) $Y \in \mathbb{R}^o$ [113]. In the classical setting, regression is used on labeled datasets that contain paired samples $\{x_i, y_i\}_{i=1}^n$, where x_i, y_i are realizations of X, Y, respectively.

Unfortunately, such an input-output correspondence is not always available in some applications. One example is flow cytometry, which is a physical experiment for measuring properties of cells, e.g., affinity to a particular target [114]. Through this process, cells are suspended in a fluid and injected into the flow cytometer, where measurements are taken using the scattering of a laser. However, the instruments are unable to differentiate the cells passing through the laser, such that the correspondence between the cell proprieties (i.e., the measurements) and the cells is unknown. This prevents us from analyzing the relationship between the instruments and the measurements using classical regression analysis, due to the missing correspondence. Another example is multi-object tracking, where we need to infer the motion of objects given consecutive frames in a video. This requires us to find the correspondence between the objects in the current frame and those in the next frame.

The two examples above can be formulated as a shuffled regression problem. Specifically, we consider a multivariate regression model

$$Y = f(X, Z; w) + \varepsilon,$$

where $X \in \mathbb{R}^d,\ Z \in \mathbb{R}^e$ are two input vectors, $Y \in \mathbb{R}^o$ is an output vector, $f : \mathbb{R}^{d+e} \to \mathbb{R}^o$ is the unknown regression model with parameters w and ε is the random noise independent on X and Z. When we sample realizations from such a regression model, the correspondence between (X, Y) and Z is not available. Accordingly, we collect two datasets $\mathcal{D}_1 = \{x_i, y_i\}_{i=1}^n$ and $\mathcal{D}_2 = \{z_j\}_{j=1}^n$, and there exists a permutation π^* such that $(x_i, z_{\pi(i)})$ corresponds to y_i in the regression model. Our goal is to recover the unknown model parameter w. Existing literature also refer to the shuffled regression problem as *unlabeled sensing*, *homomorphic sensing*, and *regression with an unknown permutation* [115]. Throughout the rest of the paper, we refer to it as *Regression WithOut Correspondence* (RWOC).

A natural choice of the objective for RWOC is to minimize the sum of squared residuals with respect to the regression model parameter w up to the permutation $\pi(\cdot)$ over the training data, i.e.,

$$\min_{w,\pi} \mathcal{L}(w, \pi) = \sum_{i=1}^{n} \| y_i - f\left(x_i, z_{\pi(i)}; w\right) \|_2^2. \tag{5.1}$$

Existing works on RWOC mostly focus on theoretical properties of the global optima to (Equation 5.1) for estimating w and π [116, 117, 118, 119, 120, 115, 121]. The development of practical algorithms, however, falls far behind from the following three aspects:

• Most of the works are only applicable to linear regression models.

• Some of the existing algorithms are of very high computational complexity, and can only handle small number of data points in low dimensions [119, 122, 123, 124]. Other algorithms choose to optimize with respect to w and π in an alteranting manner, e.g., alternating minimization in [118]. However, as there exists a strong interaction between w and π, the optimization landscape of (Equation 5.1) is ill-conditioned. Therefore, these algorithms are not effective and often get stuck in local optima.

• Most of the works only consider the case where there exists an exact one-to-one cor-

respondence between $\mathcal{D}_1$ and $\mathcal{D}_2$. For many more scenarios, however, these two datasets are not necessarily well aligned. For example, consider $\mathcal{D}_1$ and $\mathcal{D}_2$ collected from two separate databases, where the users overlap, but are not identical. As a result, there exists only partial one-to-one correspondence. A similar situation also happens to multiple-object tracking: Some objects may leave the scene in one frame, and new objects may enter the scene in subsequent frames. Therefore, not all objects in different frames can be perfectly matched. The RWOC problem with partial correspondence is known as robust-RWOC, or rRWOC [125], and is much less studied in existing literature.

To address these concerns, we propose a new computational framework – ROBOT (Regression withOut correspondence using Bilevel OptimizaTion). Specifically, we propose to formulate the regression without correspondence as a continuous optimization problem. Then by exploiting the interaction between the regression model and the data correspondence, we propose to develop a hypergradient approach based on differentiable programming techniques [86, 89]. Our hypergradient approach views the data correspondence as an operator of the regression, i.e., for a given w, the optimal correspondence is

$$\widehat{\pi}(w) = \operatorname*{argmin}_{\pi} \mathcal{L}(w, \pi). \tag{5.2}$$

Accordingly, when applying gradient descent to (Equation 5.1), we need to find the gradient with respect to w by differentiating through both the objective function $\mathcal{L}$ and the data correspondence $\widehat{\pi}(w)$. For simplicity, we refer as such a gradient to "hypergradient". Note that due to its discrete nature, $\widehat{\pi}(w)$ is actually not continuous in w. Therefore, such a hypergradient does not exist. To address this issue, we further propose to construct a smooth approximation of $\widehat{\pi}(w)$ by adding an additional regularizer to (Equation 5.2), and then we replace $\widehat{\pi}(w)$ with our proposed smooth replacement when computing the hyper gradient of w. Moreover, we also propose an efficient and scalable implementation of hypergradient computation based on simple first order algorithms and implicit differentiation, which

outperforms conventional automatic differentiation in terms of time and memory cost.

ROBOT can also be extended to the robust RWOC problem, where $\mathcal{D}_1$ and $\mathcal{D}_2$ are not necessarily exactly aligned, i.e., some data points in $\mathcal{D}_1$ may not correspond to any data point in $\mathcal{D}_2$. Specifically, we relax the constraints on the permutation $\pi(\cdot)$ [126] to automatically match related data points and ignore the unrelated ones.

At last, we conduct thorough numerical experiments to demonstrate the effectiveness of ROBOT. For RWOC (i.e., exact correspondence), we use several synthetic regression datasets and a real gated flow cytometry dataset, and we show that ROBOT outperforms baseline methods by significant margins. For robust RWOC (i.e., inexact correspondence), we consider a vision-based multiple-object tracking task, and then we show that ROBOT also achieves significant improvement over baseline methods.

Notations. Let $\|\cdot\|_2$ denote the ℓ_2 norm of vectors, $\langle\cdot,\cdot\rangle$ the inner product of matrices, i.e., $\langle A, B\rangle = \sum_{i,j} A_{ij}B_{ij}$ for matrices A and B. $a_{i:j}$ are the entries from index i to index j of vector a. Let $\mathbf{1}_n$ denote an n-dimensional vector of all ones. Denote $\frac{d(\cdot)}{d(\cdot)}$ the gradient of scalars, and $\nabla_{(\cdot)}(\cdot)$ the Jacobian of tensors. We denote $[v_1, v_2]$ the concatenation of two vectors v_1 and v_2. $\mathcal{N}(\mu, \sigma^2)$ is the Gaussian distribution with mean μ and variance σ^2.

5.2 ROBOT: A Hypergradient Approach for RWOC

We develop our hypergradient approach for RWOC. Specifically, we first introduce a continuous formulation equivalent to (Equation 5.1), and then propose a smooth bi-level relaxation with an efficient hypergradient descent algorithm.

5.2.1 Equivalent Continuous Formulation

We propose a continuous optimization problem equivalent to (Equation 5.1). Specifically, we rewrite an equivalent form of (Equation 5.1) as follows,

$$\min_{w} \min_{S\in\mathbb{R}^{n\times n}} \mathcal{L}(w, S) = \langle C(w), S\rangle \quad \text{subject to } S \in \mathcal{P}, \tag{5.3}$$

where $\mathcal{P}$ denotes the set of all $n \times n$ permutation matrices, $C(w) \in \mathbb{R}^{n \times n}$ is the loss matrix with

$$C_{ij}(w) = \| y_i - f\left(x_i, z_j; w\right) \|_2^2.$$

Note that we can relax $\mathcal{S} \in \mathcal{P}$, which is the discrete feasible set of the inner minimization problem of (Equation 5.3), to a convex set, without affecting the optimality, as suggested by the next theorem.

Proposition 3. Given any $a \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$, we define

$$\Pi(a, b) = \{A \in \mathbb{R}^{n \times m} : A\mathbf{1}_m = a, A^\top \mathbf{1}_n = b, A_{ij} \geq 0\}.$$

The optimal solution to the inner discrete minimization problem of (Equation 5.3) is also the optimal solution to the following continuous optimization problem,

$$\min_{S \in \mathbb{R}^{n \times n}} \langle C(w), S \rangle, \quad \text{s.t. } S \in \Pi(\mathbf{1}_n, \mathbf{1}_n). \tag{5.4}$$

This is a direct corollary of the Birkhoff-von Neumann theorem [127, 128], and please refer to Appendix section D.1 for more details. Theorem 3 allows us to replace $\mathcal{P}$ in (Equation 5.3) with $\Pi(\mathbf{1}_n, \mathbf{1}_n)$, which is also known as the Birkhoff polytope[1] [130]. Accordingly, we obtain the following continuous formulation,

$$\min_{w} \min_{S \in \mathbb{R}^{n \times n}} \langle C(w), S \rangle \quad \text{subject to } S \in \Pi(\mathbf{1}_n, \mathbf{1}_n). \tag{5.5}$$

Remark 1. In general, (Equation 5.3) can be solved by linear programming algorithms [131].

[1] This is a common practice in integer programming [129].

5.2.2 Conventional Wisdom: Alternating Minimization

Conventional wisdom for solving (Equation 5.5) suggests to use alternating minimization (AM, [118]). Specifically, at the k-th iteration, we first update S by solving

$$S^{(k)} = \underset{S \in \Pi(\mathbf{1}_n, \mathbf{1}_n)}{\operatorname{argmin}} \mathcal{L}(w^{(k-1)}, S),$$

and then given $S^{(k)}$, we update w using gradient descent or exact minimization, i.e.,

$$w^{(k)} = w^{(k-1)} - \eta \nabla_w \mathcal{L}(w^{(k-1)}, S^{(k)}).$$

However, AM works poorly for solving (Equation 5.5) in practice. This is because w and S have a strong interaction throughout the iterations: A slight change to w may lead to significant change to S. Therefore, the optimization landscape is ill-conditioned, and AM can easily get stuck at local optima.

5.2.3 Smooth Bi-level Relaxation

To tackle the aforementioned computational challenge, we propose a hypergradient approach, which can better handle the interaction between w and S. Specifically, we first relax (Equation 5.5) to a smooth bi-level optimization problem, and then we solve the relaxed bi-level optimization problem using the hypergradient descent algorithm.

We rewrite (Equation 5.5) as a smoothed bi-level optimization problem,

$$\min_w \mathcal{F}_\epsilon(w) = \langle C(w), S_\epsilon^*(w) \rangle, \text{ subject to } S_\epsilon^*(w) = \underset{S \in \Pi(\mathbf{1}_n, \mathbf{1}_n)}{\operatorname{argmin}} \langle C(w), S \rangle + \epsilon H(S), \quad (5.6)$$

where $H(S) = \langle \log S, S \rangle$ is the entropy of S. The regularizer $H(S)$ in (Equation 5.6)

alleviates the sensitivity of $S^*(w)$ to w. Note that if without such a regularizer, we solve

$$S^*(w) = \underset{S \in \Pi(\mathbf{1}_n, \mathbf{1}_n)}{\operatorname{argmin}} \langle C(w), S \rangle. \tag{5.7}$$

The resulting $S^*(w)$ can be discontinuous in w. This is because $S^*(w)$ is the optimal solution of a linear optimization problem, and usually lies on a vertex of $\Pi(\mathbf{1}_n, \mathbf{1}_n)$. This means that if we change w, $S^*(w)$ either stays the same or jumps to another vertex of $\Pi(\mathbf{1}_n, \mathbf{1}_n)$. The jump makes $S^*(w)$ highly sensitive to w. To alleviate this issue, we propose to smooth $S^*(w)$ by adding an entropy regularizer to the lower level problem. The entropy regularizer enforces $S^*_\epsilon(w)$ to stay in the interior of $\Pi(\mathbf{1}_n, \mathbf{1}_n)$, and $S^*_\epsilon(w)$ changes smoothly with respect to w, as suggested by the following theorem.

Theorem 5. For any $\epsilon > 0$, $S^*_\epsilon(w)$ is differentiable, if the cost $C(w)$ is differentiable with respect to w. Consequently, the objective $\mathcal{F}_\epsilon(w) = \langle C(w), S^*_\epsilon(w) \rangle$ is also differentiable.

The proof is deferred to Appendix section D.3. Note that (Equation 5.6) provides us a new perspective to interpret the relationship between w and S. As can be seen from (Equation 5.6), w and S have different priorities: w is the parameter of the leader problem, which is of the higher priority; S is the parameter of the follower problem, which is of the lower priority, and can also be viewed as an operator of w – denoted by $S^*_\epsilon(w)$. Accordingly, when we minimize (Equation 5.6) with respect to w using gradient descent, we should also differentiate through S^*_ϵ. We refer to such a gradient as "hypergradient" defined as follows,

$$\nabla_w \mathcal{F}_\epsilon(w) = \frac{\partial \mathcal{F}_\epsilon(w)}{\partial C(w)} \frac{\partial C(w)}{\partial w} + \frac{\partial \mathcal{F}_\epsilon(w)}{\partial S^*_\epsilon(w)} \frac{\partial S^*_\epsilon(w)}{\partial w} = \nabla_w \mathcal{L}(w, S) + \frac{\partial \mathcal{F}_\epsilon(w)}{\partial S^*_\epsilon(w)} \frac{\partial S^*_\epsilon(w)}{\partial w}.$$

We further examine the alternating minimization algorithm from the bi-level optimization perspective: Since $\nabla_w \mathcal{L}(w^{(k-1)}, S^{(k)})$ is not differentiable through $S^{(k)}$, AM is essentially using an inexact gradient. From a game-theoretic perspective[2], (Equation 5.6) defines a competition between the leader w and the follower S. When using AM, S only reacts

[2]The bilevel formulation can be viewed as a Stackelberg game.

to what w has responded. In contrast, when using the hypergradient approach, the leader essentially recognizes the follower's strategy and reacts to what the follower is anticipated to response through $\frac{\partial \mathcal{F}_\epsilon(w)}{\partial S^*_\epsilon(w)} \frac{\partial S^*_\epsilon(w)}{\partial w}$. In this way, we can find a better descent direction for w.

Remark 2. We use a simple example of quadratic minimization to illustrative why we expect the bilevel optimization formulation in (Equation 5.6) to enjoy a benign optimization landscape. We consider a quadratic function

$$L(a_1, a_2) = a^\top P a + b^\top a, \tag{5.8}$$

where $a_1 \in \mathbb{R}^{d_1}$, $a_2 \in \mathbb{R}^{d_2}$, $a = [a_1, a_2]$, $P \in \mathbb{R}^{(d_1+d_2)\times(d_1+d_2)}$, $b \in \mathbb{R}^{d_1+d_2}$. Let $P = \rho \mathbf{1}_{d_1+d_2} \mathbf{1}_{d_1+d_2}^\top + (1-\rho) I_{d_1+d_2}$, where $I_{d_1+d_2}$ is the identity matrix, and ρ is a constant. We solve the following bilevel optimization problem,

$$\min_{a_1} F(a_1) = L(a_1, a_2^*(a_1)) \quad \text{subject to } a_2^*(a_1) = \operatorname{argmin}_{a_2} L(a_1, a_2) + \lambda \|a_2\|_2^2, \tag{5.9}$$

where λ is a regularization coefficient. The next proposition shows that $\nabla^2 F(a_1)$ enjoys a smaller condition number than $\nabla^2_{a_1 a_1} L(a_1, a_2)$, which corresponds to the problem that AM solves.

Proposition 4. Given F defined in (Equation 5.9), we have

$$\frac{\lambda_{\max}(\nabla^2 F(a_1))}{\lambda_{\min}(\nabla^2 F(a_1))} = 1 + \frac{1-\rho+\lambda}{d_2\rho - \rho + \lambda + 1} \cdot \frac{d_1 \rho}{1-\rho} \quad \text{and} \quad \frac{\lambda_{\max}(\nabla^2_{a_1 a_1} L(a_1, a_2))}{\lambda_{\min}(\nabla^2_{a_1 a_1} L(a_1, a_2))} = 1 + \frac{d_1 \rho}{1-\rho}.$$

The proof is deferred to Appendix section D.2. As suggested by Proposition 4, $F(a_1)$ is much better-conditioned than $L(a_1, a_2)$ in terms of a_1 for high dimensional settings.

5.2.4 Solving rWOC by Hypergradient Descent

We present how to solve (Equation 5.6) using our hypergradient approach. Specifically, we compute the "hypergradient" of $\mathcal{F}_\epsilon(w)$ based on the following theorem.

Theorem 6. The gradient of $\mathcal{F}_\epsilon$ with respect to w is

$$\nabla_w \mathcal{F}_\epsilon(w) = \frac{1}{\epsilon} \sum_{i,j=1}^{n,n} \left((1 - C_{ij}) S^*_{\epsilon,ij} + \sum_{h,\ell=1}^{n,n} C_{h\ell} S^*_{\epsilon,h\ell} P_{hij} + \sum_{h,\ell=1}^{n,n} C_{h\ell} S^*_{\epsilon,h\ell} Q_{\ell ij} \right) \nabla_w C_{ij}. \tag{5.10}$$

where $\begin{bmatrix} P \\ Q \end{bmatrix} := \begin{bmatrix} -H^{-1}D \\ \mathbf{0} \end{bmatrix}$ with $P, Q \in \mathbb{R}^{n\times n\times n}, -H^{-1}D \in \mathbb{R}^{(2n-1)\times n\times n}, \mathbf{0} \in \mathbb{R}^{1\times n\times n}$,

$$D_{\ell ij} = \frac{1}{n\epsilon} \begin{cases} \delta_{\ell i} S^*_{\epsilon,ij}, & \ell = 1, \cdots, n; \\ \delta_{\ell j} S^*_{\epsilon,ij}, & \ell = n+1, \cdots, 2n-1, \end{cases} \qquad H^{-1} = -\epsilon n \begin{bmatrix} I_n + \bar{S}^*_\epsilon \mathcal{K}^{-1} \bar{S}^{*T}_\epsilon & -\bar{S}^*_\epsilon \mathcal{K}^{-1} \\ -\mathcal{K}^{-1} \bar{S}^{*T}_\epsilon & \mathcal{K}^{-1} \end{bmatrix},$$

and $\mathcal{K} = I_{n-1} - \bar{S}^{*T}_\epsilon \bar{S}^*_\epsilon, \quad \bar{S}^*_\epsilon = S^*_{\epsilon,1:n,1:n-1}$.

The proof is deferred to Appendix section D.3. Theorem Theorem 6 suggests that we first solve the lower level problem in (Equation 5.6),

$$S^*_\epsilon = \underset{S\in\Pi(\mathbf{1}_n,\mathbf{1}_n)}{\operatorname{argmin}} \langle C(w), S\rangle + \epsilon H(S), \tag{5.11}$$

and then substitute S^*_ϵ into (Equation 5.10) to obtain $\nabla_w \mathcal{F}_\epsilon(w)$.

Note that the optimization problem in (Equation 5.11) can be efficiently solved by a variant of Sinkhorn algorithm [2, 3]. Specifically, (Equation 5.11) can be formulated as an entropic optimal transport (EOT) problem [1, 90], which aims to find the optimal way to transport the mass from a categorical distribution with weight $\mu = [\mu_1, \ldots, \mu_n]^\top$ to another categorical distribution with weight $\nu = [\nu_1, \ldots, \nu_m]^\top$,

$$\begin{aligned} \Gamma^* &= \underset{\Gamma\in\Pi(\mu,\nu)}{\operatorname{argmin}} \langle M, \Gamma\rangle + \epsilon H(\Gamma), \\ \text{with } \Pi(\mu,\nu) &= \{\Gamma \in \mathbb{R}^{n\times m} : \Gamma \mathbf{1}_m = \mu, \Gamma^\top \mathbf{1}_n = \nu, \Gamma_{ij} \geq 0\}, \end{aligned} \tag{5.12}$$

where $M \in \mathbb{R}^{n\times m}$ is the cost matrix with M_{ij} the transport cost. When we set the two categorical distributions as the empirical distribution of $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively,

$$M = C(w) \quad \text{and} \quad \mu = \nu = \mathbf{1}_n/n,$$

one can verify that (Equation 5.12) is a scaled lower problem of (Equation 5.6), and their optimal solutions satisfies $S^*_\epsilon = n\Gamma^*$. Therefore, we can apply Sinkhorn algorithm to solve the EOT problem in (Equation 5.12): At the ℓ-th iteration, we take

$$p^{(\ell+1)} = \frac{\mu}{Gq^{(\ell)}} \text{ and } q^{(\ell+1)} = \frac{\nu}{G^\top p^{(\ell+1)}}, \text{ where } q^{(0)} = \frac{1}{n}\mathbf{1}_n \text{ and } G_{ij} = \exp\left(\frac{-C_{ij}(w)}{\epsilon}\right),$$

$G \in \mathbb{R}^{n\times n}$, and the division here is entrywise. Let p^* and q^* denote the stationary points. Then we obtain $S^*_{\epsilon,ij} = np^*_i G_{ij} q^*_j$.

Remark 3. The Sinkhorn algorithm is iterative and cannot exactly solve (Equation 5.11) within finite steps. As the Sinkhorn algorithm is very efficient and attains linear convergence, it suffices to well approximate the gradient $\nabla_w \mathcal{F}_\epsilon(w)$ using the output inexact solution.

5.3 ROBOT for Robust Correspondence

We next propose a robust version of ROBOT to solve rRWOC [125]. Note that in (Equation 5.6), the constraint $S \in \Pi(\mathbf{1}_n, \mathbf{1}_n)$ enforces a one-to-one matching between $\mathcal{D}_1$ and $\mathcal{D}_2$. For rRWOC, however, such an exact matching may not exist. For example, we have $n < m$, where $n = |\mathcal{D}_1|$, $m = |\mathcal{D}_2|$. Therefore, we need to relax the constraint on S.

Motivated by the connection between (Equation 5.6) and (Equation 5.12), we propose

to solve the following lower problem[3],

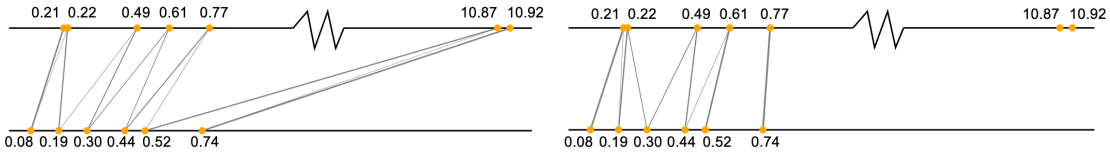
$$(S_{\mathrm{r}}^*(w), \bar{\mu}^*, \bar{\nu}^*) = \underset{S \in \Pi(\bar{\mu}, \bar{\nu})}{\operatorname{argmin}} \langle C(w), S \rangle + \epsilon H(S), \tag{5.13}$$
$$\text{subject to } \bar{\mu}^\top \mathbf{1}_n = n,\ \bar{\nu}^\top \mathbf{1}_m = m,\ \|\bar{\mu} - \mathbf{1}_n\|_2^2 \leq \rho_1,\ \|\bar{\nu} - \mathbf{1}_m\|_2^2 \leq \rho_2,$$

where $S_{\mathrm{r}}^*(w) \in \mathbb{R}^{n \times m}$ denotes an inexact correspondence between $\mathcal{D}_1$ and $\mathcal{D}_2$. As can be seen in (Equation 5.13), we relax the marginal constraint $\Pi(\mathbf{1}, \mathbf{1})$ in (Equation 5.6) to $\Pi(\bar{\mu}, \bar{\nu})$, where $\bar{\mu}, \bar{\nu}$ are required to not deviate much from $\mathbf{1}$. Problem (Equation 5.13) relaxes the marginal constraints $\Pi(\mathbf{1}, \mathbf{1})$ in the original problem to $\Pi(\bar{\mu}, \bar{\nu})$, where $\bar{\mu}, \bar{\nu}$ are picked such that they do not deviate too much from $\mathbf{1}$. Illustrative examples of the exact and robust alignments are provided in Figure Figure 5.1.

Computationally, (Equation 5.13) can be solved by taking the Sinkhorn iteration and the projected gradient iteration in an alternating manner (See more details in Appendix section D.4). Given $S_{\mathrm{r}}^*(w)$, we solve the upper level optimization in (Equation 5.6) to obtain w^*, i.e.,

$$w^* = \underset{w}{\operatorname{argmin}} \langle C(w), S_{\mathrm{r}}^*(w) \rangle.$$

Similar to the previous section, we use a first-order algorithm to solve this problem, and we derive explicit expressions for the update rules. See Appendix section D.5 for details.



(a) Original (b) Robust

Figure 5.1: *Illustrative example of exact (L) and robust (R) alignments.* The robust alignment can drop potential outliers and only match data points close to each other.

[3]The idea is inspired by the marginal relaxation of optimal transport, first independently proposed by [132] and [133], and later developed by [134] and [126]. [135] share the same formulation as ours.

5.4 Experiment

We evaluate ROBOT and ROBOT-robust on both synthetic and real-world datasets, including flow cytometry and multi-object tracking. We first present numerical results and then we provide insights in the discussion section. Experiment details and auxiliary results can be found in Appendix section D.6.

5.4.1 Unlabeled Sensing

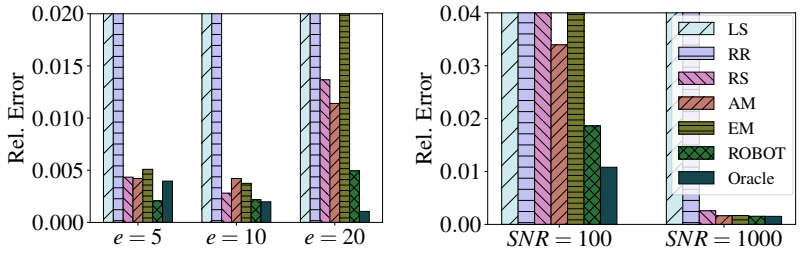
Data Generation. We follow the unlabeled sensing setting [121] and generate $n = 1000$ data points $\{(y_i, z_i)\}_{i=1}^n$, where $z_i \in \mathbb{R}^e$. Note here we take $d = 0$. We first generate $z_i, w \sim \mathcal{N}(\mathbf{0}_e, \boldsymbol{I}_e)$, and $\varepsilon_i \sim \mathcal{N}(0, \rho^2_{\text{noise}})$. Then we compute $y_i = z_i^\top w + \varepsilon_i$. We randomly permute the order of 50% of z_i so that we lose the Z-to-Y correspondence. We generate the test set in the same way, only without permutation.

Baselines and Training. We consider the following scalable methods:

1. *Oracle*: Standard linear regression where no data are permuted.
2. *Least Squares* (*LS*): Standard linear regression, i.e., treating the data as if they are not permuted.
3. *Alternating Minimization* (*AM*, [118]): We iteratively solve the correspondence given w, and update w using gradient descent with the correspondence.
4. *Stochastic EM* [114]: A stochastic EM approach to recover the permutation.
5. *Robust Regression* (*RR*, [136, 137]). A two-stage block coordinate descent approach to discard outliers and fit regression models.
6. *Random Sample* (*RS*, [125]): A random sample consensus (RANSAC) approach to estimate w.

We initialize AM, EM and ROBOT using the output of RS with multi-start. We adopt a linear model $f(Z; w) = Z^\top w$. Models are evaluated by the relative error on the test set,

i.e., error $= \sum_i (\widehat{y}_i - y_i)^2 / \sum_i (y_i - \bar{y})^2$, where $\widehat{y}_i$ is the predicted label, and $\bar{y}$ is the mean of $\{y_i\}$.



(a) $\rho^2_{\text{noise}} = 0.1$ (b) $e = 10$

Figure 5.2: *Unlabeled sensing. Results are the mean over 10 runs.* $\texttt{SNR} = \|w\|_2^2/\rho^2_{\text{noise}}$ *is the signal-to-noise ratio.*

Results. We visualize the results in Figure Figure 5.2. In all the experiments, ROBOT achieves better results than the baselines. Note that the relative error is larger for all methods except Oracle as the dimension and the noise increase. For low dimensional data, e.g., $e = 5$, our model achieves even better performance than Oracle. We include more discussions on using RS as initializations in Section section 5.5.

5.4.2 Nonlinear Regression

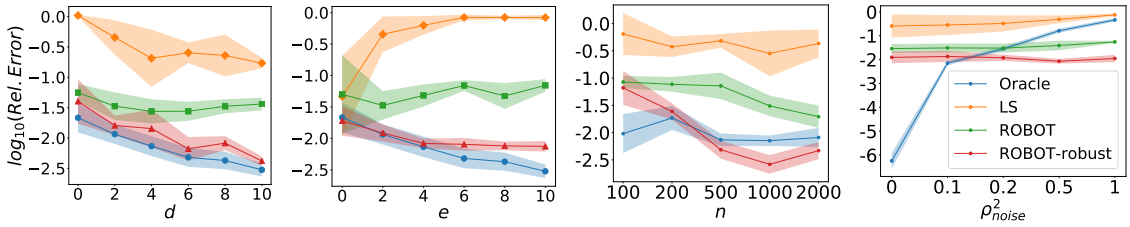


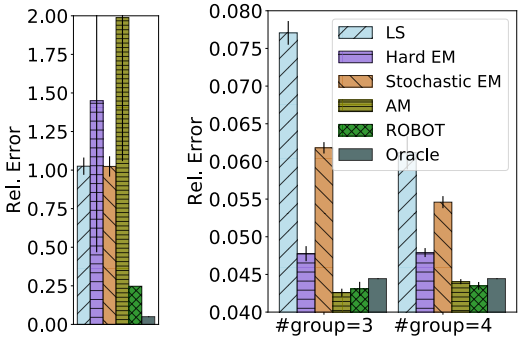
Figure 5.3: *Nonlinear regression. We use* $n = 1000$, $d = 2$, $e = 3$, $\rho^2_{\text{noise}} = 0.1$ *as defaults.*

Data Generation. We mimic the scenario where the dataset is collected from different platforms. Specifically, we generate n data points $\{(y_i, [x_i, z_i])\}_{i=1}^n$, where $x_i \in \mathbb{R}^d$ and $z_i \in \mathbb{R}^e$. We first generate $x_i \sim \mathcal{N}(\mathbf{0}_d, \boldsymbol{I}_d)$, $z_i \sim \mathcal{N}(\mathbf{0}_e, \boldsymbol{I}_e)$, $w \sim \mathcal{N}(\mathbf{0}_{d+e}, \boldsymbol{I}_{d+e})$, and $\varepsilon_i \sim \mathcal{N}(0, \rho^2_{\text{noise}})$. Then we compute $y_i = f([x_i, z_i]; w) + \varepsilon_i$. Next, we randomly permute the order of $\{z_i\}$ so that we lose the data correspondence. Here, $\mathcal{D}_1 = \{(x_i, y_i)\}$ and

$\mathcal{D}_2 = \{z_j\}$ mimic two parts of data collected from two separate platforms. Since we are interested in the response on platform one, we treat all data from platform two, i.e., $\mathcal{D}_2$, as well as 80% of data in $\mathcal{D}_1$ as the training data. The remaining data from $\mathcal{D}_1$ are the test data. Notice that we have different number of data on $\mathcal{D}_1$ and $\mathcal{D}_2$, i.e., the correspondence is not exactly one-to-one.

Baselines and Training. We consider a nonlinear function $f(X, Z; w) = \sum_{k=1}^{d} \sin([X, Z]_k w_k)$. In this case, we consider only two baselines — Oracle and LS, since the other baselines in the previous section are designed for linear models. We evaluate the regression models by the transport cost divided by $\sum_i (y_i - \bar{y})^2$ on the test set.

Results. As shown in Figure Figure 5.3, ROBOT-robust consistently outperforms ROBOT and LS, demonstrating the effectiveness of our robust formulation. Moreover, ROBOT-robust achieves better performance than Oracle when the number of training data is large or when the noise level is high.



(a) *FC* (b) *GFC*

Figure 5.4: *Relative error of different methods.*

5.4.3 Flow Cytometry

In flow cytometry (FC), a sample containing particles is suspended in a fluid and injected into the flow cytometer, but the measuring instruments are unable to preserve the correspondence between the particles and the measurements. Different from FC, gated flow

Table 5.1: *Experiment results on MOT. Here, ↑ suggests the larger the better, and ↓ suggests the smaller the better.*

Data	Method	MOTA↑	MOTP↑	IDF1↑	MT↑	ML↓	FP↓	FN↓	IDS↓
MOT17 (train)	ROBOT	**48.3**	82.6	55.3	407	553	22,443	149,988	1,811
	w/o ROBOT	44.0	81.3	49.9	404	550	36,187	149,131	3,204
MOT17 (dev)	ROBOT	**48.2**	76.6	43.4	455	904	29,419	259,714	3,228
	w/o ROBOT	42.1	75.0	36.8	414	890	61,210	259,318	6,138
	SORT	43.1	77.8	39.8	295	997	28,398	287,582	4,852
MOT20 (train)	ROBOT	**56.2**	84.9	47.6	805	288	113,752	377,247	5,888
	w/o ROBOT	48.8	81.5	40.2	769	290	186,245	384,562	10,153
MOT20 (dev)	ROBOT	**45.0**	76.9	34.0	394	257	70,416	210,425	3,683
	w/o ROBOT	38.5	75.1	27.0	383	233	104,958	207,627	5,696
	SORT	42.7	78.5	45.1	208	326	27,521	264,694	4,470

cytometry (GFC) uses "gates" to sort the particles into one of many bins, which provides partial ordering information since the measurements are provided individually for each bin. In practice, there are usually 3 or 4 bins.

Settings. We adopt the dataset from [138]. Following [118], the outputs y_i's are normalized, and we select the top 20 significant features by a linear regression on the top 1400 items in the dataset. We use 90% of the data as the training data, and the remaining as test data. For ordinary FC, we randomly shuffle all the labels in the training set. For GFC, the training set is first sorted by the labels, and then divided into equal-sized groups, mimicking the sorting by gates process. The labels in each group are then randomly shuffled. To simulate gating error, 1% of the data are shuffled across the groups. We compare ROBOT with Oracle, LS, Hard EM (a variant of Stochastic EM proposed in [114]), Stochastic EM, and AM. We use relative error on the test set as the evaluation metric.

Results. As shown in Figure Figure 5.4, while AM achieves good performance on GFC when the number of groups is 3, it behaves poorly on the FC task. ROBOT, on the other hand, is efficient on both tasks.

Figure 5.5: *One frame in* `MOT20` *with detected bounding boxes in yellow.*

5.4.4 Multi-Object Tracking

In this section we extend our method to vision-based Multi-Object Tracking (MOT), a task with broad applications in mobile robotics and autonomous driving. Given a video and the current frame, the goal of MOT is to predict the locations of the objects in the next frame. Specifically, object detectors [139, 140] first provide us the potential locations of the objects by their bounding boxes. Then, MOT aims to assign the bounding boxes to trajectories that describe the path of individual objects over time. Here, we formulate the current frame and the objects' locations in the current frame as $\mathcal{D}_2 = \{z_j\}$, while we treat the next frame and the locations in the next frame as $\mathcal{D}_1 = \{(x_i, y_i)\}$.

Existing deep learning based MOT algorithms require large amounts of annotated data, i.e., the ground truth of the correspondence, during training. Different from them, our algorithm does not require the correspondence between $\mathcal{D}_1$ and $\mathcal{D}_2$, and all we need is the video. This task is referred to as *unsupervised MOT* [141].

Related Works. To the best of our knowledge, the only method that accomplishes unsupervised end-to-end learning of MOT is [141]. However, it targets tracking with low densities, e.g., `Sprites-MOT`, which is different from our focus.

Settings. We adopt the `MOT17` [142] and the `MOT20` [143] datasets. Scene densities of the two datasets are 31.8 and 170.9, respectively, which means the scenes are pretty crowded as we illustrated in Figure Figure 5.5. We adopt the DPM detector [139] on `MOT17` and the Faster-RCNN detector [140] on `MOT20` to provide us the bounding boxes. Inspired by [144], the cost matrix is computed as the average of the Euclidean center-point distance and the Jaccard distance between the bounding boxes,

$$C_{ij}(w) = \frac{1}{2}\left(\frac{\|c(f(z_j;w)) - c(y_i)\|_2}{\sqrt{H^2+W^2}} + \mathcal{J}(f(z_j;w), y_i)\right),$$

where $c(\cdot)$ is the location of the box center, H and W are the height and the width of the video frame, and $\mathcal{J}(\cdot,\cdot)$ is the Jaccard distance defined as 1-IoU (Intersection-over-Union). We utilize the single-object tracking model SiamRPN[4] [145] as our regression model f. We apply ROBOT-robust with $\rho_1 = \rho_2 = 10^{-3}$. See Appendix section D.6 for more detailed settings.

Results. We demonstrate the experiment results in Table Table 5.1, where the evaluation metrics follow [146]. In the table, $\uparrow$ represents the higher the better, and $\downarrow$ represents the lower the better. ROBOT signifies the model trained by ROBOT-robust, and w/o ROBOT means the pretrained model in [145]. The scores are improved significantly after training with ROBOT-robust.

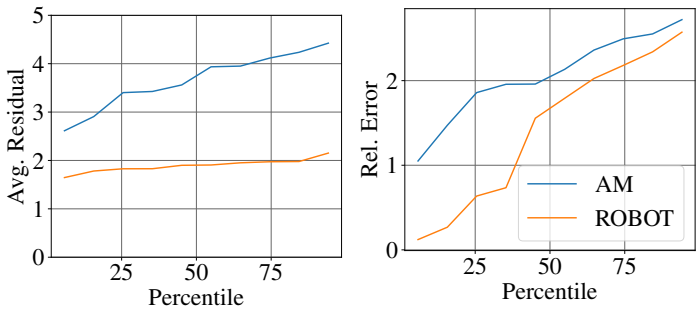
We also include the scores of the SORT model [147] obtained from the dataset platform. Different from SiamRPN and SiamRPN+ROBOT, SORT is a supervised learning model. As shown, our unsupervised training framework achieves comparable or even better performance.

[4]The initial weights of f are obtained from `https://github.com/foolwood/DaSiamRPN`.

5.5 Discussion

• **Sensitivity to initialization**. As stated in [117], obtaining the global optima of (Equation 5.1) is in general an NP-hard problem. Some "global" methods methods use global optimization techniques and have exponential complexity, e.g., [119], which is not applicable to large data. The other "local" methods only guarantee converge to local optima, and the convergence is very sensitive to initialization. Compared with existing "local" methods, our method is computationally efficient and greatly reduces the sensitivity to initialization.

To demonstrate such an advantage, we run AM and ROBOT with 10 different initial solutions, and then we sort the results based on (a) the averaged residual on the training set, and (b) the relative prediction error on the test set. We plot the percentiles in Figure Figure 5.6. Here we use fully shuffled data under the unlabeled sensing setting, and we set $n = 1000$, $d = 5$, $\rho^2_{\text{noise}} = 0.1$, and $\epsilon = 10^{-2}$. We can see that ROBOT is able to find "good" solutions in 30% of the cases (The relative prediction error is smaller than 1), but AM is more sensitive to the initialization and cannot find "good" solutions.

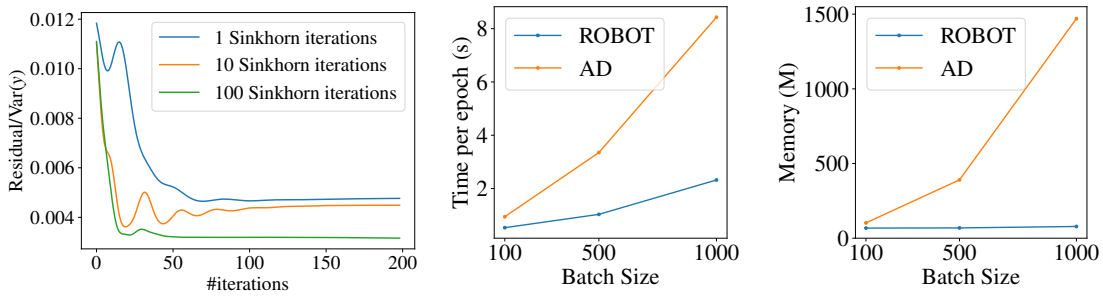


(a) Training residual (b) Test error

Figure 5.6: ROBOT and AM with *different initial solutions.*

• **ROBOT v.s. Automatic Differentiation (AD).** Our algorithm computes the Jacobian matrix directly based on the KKT condition of the lower problem (Equation 5.11). An alternative approach to approximate the Jacobian is the automatic differentiation through the Sinkhorn iterations for updating S when solving (Equation 5.11). As suggested by Fig-

ure Figure 5.7 (a), running Sinkhorn iterations until convergence (200 Sinkhorn iterations) can lead to a better solution[5]. In order to apply AD, we need to store all the intermediate updates of all the Sinkhorn iterations. This require the memory usage to be proportional to the number of iterations, which is not necessarily affordable. In contrast, applying our explicit expression for the backward pass is memory-efficient. Moreover, we also observe that AD is much more time-consuming than our method. The timing performance and memory usage are shown in Figure Figure 5.7 (b)(c), where we set $n = 1000$.



(a) (b) (c)

Figure 5.7: *The comparisons to AD. (a) Convergence under different number of Sinkhorn iterations of AD. (b) Time comparison. (c) Memory comparison.*

• **Connection to EM.** [114] adopt an Expectation Maximization (EM) method for RWOC, where S is modeled as a latent random variable. Then in the M-step, one maximizes the expected likelihood of the data over S. This method shares the same spirit as ours: We avoid updating w using one single permutation matrix like AM. However, this method is very dependent on a good initialization. Specifically, if we randomly initialize w, the posterior distribution of S in this iteration would be close to its prior, which is a uniform distribution. In this way, the follow-up update for w is not informative. Therefore, the solution of EM would quickly converge to an undesired stationary point. Figure Figure 5.8 illustrates an example of converged correspondence, where we adopt $n = 30, o = e = 1, d = 0$. For this reason, we initialize EM with good initial points, either by RS or AM throughout all experiments.

[5]We remark that running one iteration sometimes cannot converge.

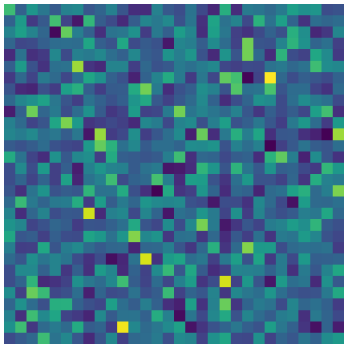


Figure 5.8: *Expected correspondence in EM.*

Table 5.2: *Pairwise comparisons between RS alone and the combination of RS and ROBOT. The relative error ratio is the ratio of the relative errors of RS alone and RS+ROBOT combination. Ratios larger than* 1 *suggest that RS performs worse than RS+ROBOT combination.*

Proportion	25%	50%	75%
Rel. error ratio	1.04 ± 0.20	1.29 ± 0.32	1.27 ± 0.34

• **Combination with RS**. As suggested in Figure Figure 5.2, although RS cannot perform well itself, retraining the output of RS using our algorithms increases the performance by a large margin. To show that combining RS and ROBOT can achieve better results than RS alone, we compare the following two cases: i). Subsample 2×10^5 times using RS; ii). Subsample 10^5 times using RS followed by ROBOT for 50 training steps. The result is shown in Table Table 5.2. For a larger permutation proportion, RS alone cannot perform as well as RS+ROBOT combination. Here, we have 10 runs for each proportion. We adopt $\texttt{SNR}= 100$, $d = 5$ for data, and $\epsilon = 10^{-4}$, learning rate 10^{-4} for ROBOT training.

• **Related works with additional constraints.** There is another line of research which improves the computational efficiency by solving variants of RWOC with additional constraints. Specifically, [148, 149] assume an isotonic function (note that such an assumption may not hold in practice), and [150, 136, 137, 151, 125] assume only a small fraction of the correspondence is missing. Our method is also applicable to these problems, as long as the additional constraints can be adapted to the implicit differentiation step.

• **More applications of RWOC.** RWOC problems generally appear for two reasons. First, the measuring instruments are unable to preserve the correspondence. In addition to GFC and MOT, we list a few more examples: SLAM tracking [152], archaeological measurements [153], large sensor networks [154], pose and correspondence estimation [155], and the genome assembly problem from shotgun reads [156]. Second, the data correspondence is masked for privacy reasons. For example, we want to build a recommender system for a new platform, borrowing user data from a mature platform.

Appendices

APPENDIX A

APPENDIX ON IPOT

A.1 More Analysis on IPOT

A.1.1 Convergence w.r.t. L

As mentioned in Section 6.1, we provide the test result of 64D Gaussian distributed data here. We choose the computed Wasserstein distance $\langle \mathbf{\Gamma}, \boldsymbol{C} \rangle$ as the indicator of convergence, because while the optimal transport plan might not be unique, the computed Wasserstein distance at convergence must be unique and minimized to ground truth. We use the empirical distribution as input distributions, i.e.,

$$\begin{aligned} &W(\{x_i\}, \{g_\theta(z_j)\}) = \min_{\mathbf{\Gamma}} \langle \boldsymbol{C}(\theta), \mathbf{\Gamma} \rangle \\ &\text{s.t. } \mathbf{\Gamma}\mathbf{1}_n = \frac{1}{n}\mathbf{1}_n, \mathbf{\Gamma}^T\mathbf{1}_n = \frac{1}{n}\mathbf{1}_n. \end{aligned} \tag{A.1}$$

As shown in Figure Figure A.1, the convergence rate is also linear. For comparison, we also provide the convergence path of Sinkhorn iteration. The result cannot converge to ground truth because the method is essentially regularized.

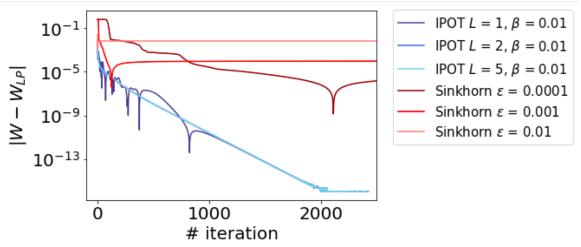


Figure A.1: The plot of differences in computed Wasserstein distances w.r.t. number of iterations for 64D Gaussian distributed data. Here, W are the Wasserstein distance computed at current iteration. W_{LP} is computed by simplex method, and is used as ground truth. The test adopts $c(x, y) = ||x - y||_2$. Due to random data is used, the number of iteration that the algorithm reaches 10^{-17} varies from 1000 to around 5000 according to our tests.

Remark. When we are talking about amount of regularization, usually we are referring to the magnitude of ϵ for Sinkhorn, or the equivalent magnitude of ϵ computed from remark in Section 3 for IPOT method. However, the amount of regularization in a loss function should be quantified by $\epsilon/||\boldsymbol{C}||$, instead of ϵ alone. That is why in this paper, different magnitude of ϵ is used for different application.

A.1.2 How IPOT Avoids Instability

Heuristically, if Sinkhorn does not underflow, with enough iteration, the result of IPOT is approximately the same as Sinkhorn with $\epsilon^{(t)} = \beta/t$. The difference lies in IPOT is a principled way to avoid underflow and can converge to arbitrarily small regularization, while Sinkhorn always causes numerical difficulty when $\epsilon \to 0$, even with scheduled decreasing ϵ like [5]. More specifically, in IPOT, we can factor $\Gamma = \text{diag}(\boldsymbol{u}_1)\boldsymbol{G}^t\text{diag}(\boldsymbol{u}_2)$, where $(\cdot)^t$ is element-wise exponent operation, and $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ are two scaling vectors. So we have $\epsilon^{(t)} = \beta/t$. As t goes infinity, all entries of $\boldsymbol{G}^t$ would underflow if we use Sinkhorn with $\epsilon^{(t)} = \beta/t$. But we know $\boldsymbol{\Gamma}^*$ is neither all zeros nor contains infinity. So instead of computing $\boldsymbol{G}^t$, $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ directly, we use $\boldsymbol{\Gamma}^t$ to record the multiplication of $\boldsymbol{G}^t$ with part of $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ in each step, so the entries of $\boldsymbol{\Gamma}^t$ will not over/underflow. The explicit computation of $\boldsymbol{G}^t$ is not needed.

Therefore, by tuning β and iteration number, we can achieve the result of arbitrary amount of regularization with IPOT.

A.2 Learning Generative Models

In this section, we show the derivation for the learning algorithm, and more tests result.

For simplicity, we assume $|\{x_i\}| = |\{z_j\}| = n$. Given a dataset $\{x_i\}$ and some noise $\{z_j\}$ [157, 45], our goal is to find a parameterized function $g_\theta(\cdot)$ that minimize

$W(\{x_i\}, \{g_\theta(z_j)\})$,

$$\begin{aligned} W(\{x_i\}, \{g_\theta(z_j)\}) = \ & \min_{\mathbf{\Gamma}} \langle \boldsymbol{C}(\theta), \mathbf{\Gamma} \rangle \\ & \text{s.t. } \mathbf{\Gamma}\mathbf{1}_n = \frac{1}{n}\mathbf{1}_n, \mathbf{\Gamma}^T\mathbf{1}_n = \frac{1}{n}\mathbf{1}_n, \end{aligned} \tag{A.2}$$

where $\boldsymbol{C}(\theta) = [c(x_i, g_\theta(z_j))]$.Usually, g_θ is parameterized by a neural network with parameter θ, and the minimization over θ is done by stochastic gradient descent.

In particular, given current estimation θ, we can obtain optimum $\mathbf{\Gamma}^*$ by IPOT, and compute the Wasserstein distance by $\langle \boldsymbol{C}(\theta), \mathbf{\Gamma}^* \rangle$ accordingly. Then, we can further update θ by the gradient of current Wasserstein distance. There are two ways to solve the gradient: One is auto-diff based method such as [8], the other is based on the envelope theorem [16]. Different from the auto-diff based methods, the back-propagation based on envelope theorem does not go into proximal point iterations because the derivative over $\mathbf{\Gamma}^*$ is not needed, which accelerates the learning process greatly. This also has significant implications numerically because the derivative of a computed quantity tends to amplify the error. Therefore, we adopt envelope based method.

Theorem 7. Envelope theorem. Let $f(x, \theta)$ and $l(x)$ be real-valued continuously differentiable functions, where $x \in \mathbb{R}^n$ are choice variables and $\theta \in \mathbb{R}^m$ are parameters. Denote x^* to be the optimal solution of f with constraint $l = 0$ and fixed θ, i.e.

$$x^* = \underset{x}{\operatorname{argmin}} f(x, \theta) \quad s.t. \quad l(x) = 0.$$

Then, assume that V is continuously differentiable function defined as $V(\theta) \equiv f(x^*(\theta), \theta)$, the derivative of V over parameters is

$$\frac{\partial V(\theta)}{\partial \theta} = \frac{\partial f}{\partial \theta}.$$

In our case, because $\mathbf{\Gamma}^*$ is the minimization of $\langle \mathbf{\Gamma}, \boldsymbol{C}(\theta) \rangle$ with constraints, we have

$$\frac{\partial W(\{x_i\}, \{g_\theta(z_j)\})}{\partial \theta} = \frac{\partial \langle \mathbf{\Gamma}^*, \boldsymbol{C}(\theta) \rangle}{\partial \theta}$$
$$= \langle \mathbf{\Gamma}^*, \frac{\partial \boldsymbol{C}(\theta)}{\partial \theta} \rangle = \langle \mathbf{\Gamma}^*, 2(g_\theta(z_j) - x_i) \frac{\partial g_\theta(z_i)}{\partial \theta} \rangle,$$

where we assume $C_{ij}(\theta) = \|x_i - g_\theta(z_j)\|_2^2$, but the algorithm can also adopt other metrics. The derivation is in supplementary materials. The flowchart is shown in Figure Figure A.2, and the algorithm is shown in Algorithm Algorithm 6.

Note Sinkhorn distance is defined as $S(\{x_i\}, \{g_\theta(z_j)\}) = \langle \boldsymbol{C}(\theta), \mathbf{\Gamma}^* \rangle$, where $\mathbf{\Gamma}^* = \operatorname{argmin}_{\mathbf{\Gamma} \in \Sigma(\mathbf{1}/n, \mathbf{1}/n)} \langle \boldsymbol{C}(\theta), \mathbf{\Gamma} \rangle + \epsilon h(\mathbf{\Gamma})$. If Sinkhorn distance is used in learning generative models, envelope theorem cannot be used because the loss function for optimizing θ and $\mathbf{\Gamma}$ is not the same.

In the tests, we observe the method in [8] suffers from shrinkage problem, i.e. the generated distribution tends to shrink towards the target mean. The recovery of target distribution is sensitive to the weight of regularization term ϵ. Only relatively small ϵ can lead to a reasonable generated distribution.

Algorithm 6 Learning generative networks

Input: real data $\{x_i\}$, initialized generator g_θ

while not converged **do**

Sample a batch of real data $\{x_i\}_{i=1}^n$

Sample a batch of noise data $\{z_j\}_{i=1}^n \sim q$

$C_{ij} := c(x_i, g_\theta(z_j)) := ||x_i - g_\theta(z_j)||_2^2$

$\mathbf{\Gamma}$ = IPOT($\frac{1}{n}\mathbf{1}_n, \frac{1}{n}\mathbf{1}_n, \boldsymbol{C}$)

Update θ with $\langle \mathbf{\Gamma}, [2(x_i - g_\theta(z_j)) \frac{\partial g_\theta(z_j)}{\partial \theta}] \rangle$

end while

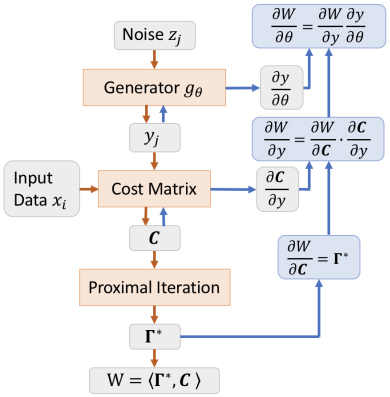


Figure A.2: The architecture of the learning model using Envelope theorem in detail. According to Envelope theorem, we do not need to compute $\frac{\partial W}{\partial \Gamma^*}$, so we do not need to back-propagate into the iteration.

A.2.1 Synthetic Test

In section 5.1, we show the learning result of Sinkhorn and IPOT in 2D case. In Figure Figure A.3 we show sequences of results for a 1D-1D generator, respectively. The upper sequence is IPOT with $\beta = 0.01, 0.025, 0.05, 0.075, 0.1$. The results barely change w.r.t. β. The lower sequence is the corresponding Sinkhorn results. The results shrink to the mean of target data, as expected. Also, we observe the learned distribution tends to have a tail that is not in the range of target data (also in 2D result, we do not include that part for a better view). It might be because the range of support that has a small probability has very small gradient when updated. Once the distribution is initialized to have a tail with small probability, it can hardly be updated. But this theory cannot explain why larger ϵ corresponds to longer tails. The tails can be on the left or right. We pick the ones on the left for easier comparison.

A.2.2 MNIST Test

The same shrinkage can be observed in MNIST data as well. See figure Figure A.4. While $\epsilon = 0.1$ covers most shapes of the numbers, $\epsilon = 1$ only covers a fraction, and $\epsilon = 10$ seems to cover only the mean of images.

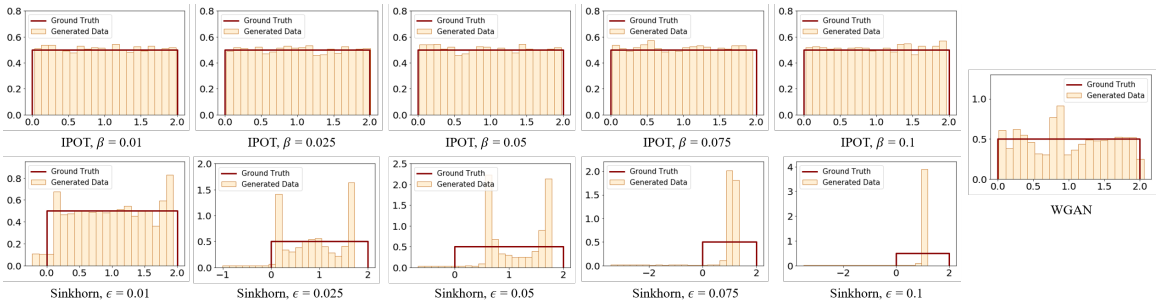
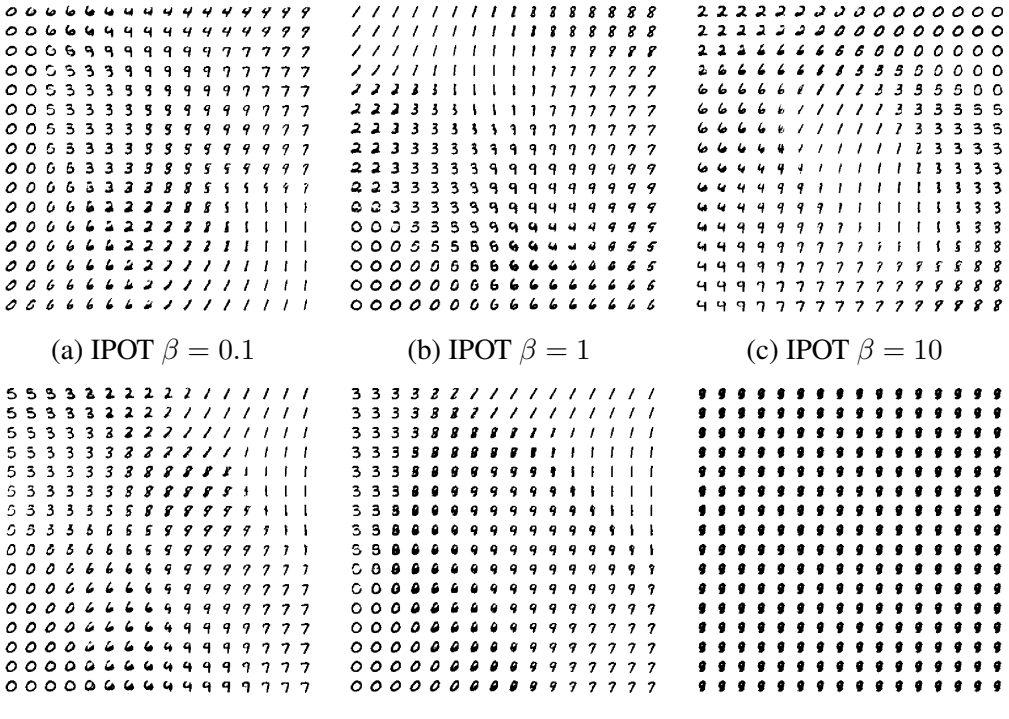


Figure A.3: The sequences of learning result of IPOT, Sinkhorn. In each figure, the orange histogram is the histogram of generated data, while the red line represents the PDF of the ground truth of target distribution.



(a) IPOT $\beta = 0.1$ (b) IPOT $\beta = 1$ (c) IPOT $\beta = 10$

(d) Sinkhorn $\epsilon = 0.1$ (e) Sinkhorn $\epsilon = 1$ (f) Sinkhorn $\epsilon = 10$

Figure A.4: Plots of MNIST learning result under comparable resources with different ϵ. They both use batch size=200, number of hidden layer=1, number of nodes of hidden layer=500, number of iteration=500, learning rate = 10^{-4}. Note that despite we show result of $\epsilon = 0.1$ here, the algorithm does not run stably. It would sometimes fail due to numerical issue.

A.3 General Bregman Proximal Point Algorithm

In the main body of the paper, we discussed the proximal point algorithm with specific Bregman distance, which is generated through the traditional entropy function. In this section, we generalize our results by proving the effectiveness of proximal point algorithm with general Bregman distance. Bregman distance is applied to measure the discrepancy between different matrices which turns out to be one of the key ideas in regularized optimal transport problems. Its special structure also give rise to proximal-type algorithms and projectors in solving optimization problems.

A.3.1 Basic Algorithm Framework and Preliminaries

The fundamental iterative scheme of general Bregman proximal point algorithm can be denoted as

$$x^{(t+1)} = \arg\min_{x \in X} \left\{ f(x) + \beta^{(t)} D_h(x, x^{(t)}) \right\}, \tag{A.3}$$

where $t \in \mathbb{N}$ is the index of iteration, and $D_h(x, x^{(t)})$ denotes a general Bregman distance between x and $x^{(t)}$ based on a Legendre function h (The definition is presented in the following). In the main body of the paper, h is specialized as the classical entropy function and as follows the related Bregman distance reduces to the generalized KL divergence. Furthermore, the Sinkhorn-Knopp projection can be introduced to compute each iterative subproblem. In the following, we present some fundamental definitions and lemmas.

Definition 1. Legendre function: Let $h : X \to (-\infty, \infty]$ be a lsc proper convex function. It is called

1. *Essentially smooth*: if h is differentiable on $\operatorname{int}\operatorname{dom} h$, with moreover $\|\nabla h(x^{(t)})\| \to \infty$ for every sequence $\{x^{(t)}\} \subset \operatorname{int}\operatorname{dom} h$ converging to a boundary point of $\operatorname{dom} h$ as $t \to +\infty$;

2. *Legendre type*: if h is essentially smooth and strictly convex on $\operatorname{int}\operatorname{dom} h$.

Definition 2. Bregman distance: any given Legendre function h,

$$D_h(x,y) = h(x) - h(y) - \langle \nabla h(y), x - y \rangle, \quad \forall x \in \operatorname{dom} h, \forall y \in \operatorname{int} \operatorname{dom} h, \tag{A.4}$$

where D_h is strictly convex with respect to its first argument. Moreover, $D_h(x,y) \geq 0$ for all $(x,y) \in \operatorname{dom} h \times \operatorname{int} \operatorname{dom} h$, and it is equal to zero if and only if $x = y$. However, D_h is in general asymmetric, i.e., $D_h(x,y) \neq D_h(y,x)$.

Definition 3. Symmetry Coefficient: Given a Legendre function $h : X \to (-\infty, \infty]$, its symmetry coefficient is defined by

$$\alpha(h) = \inf \left\{ \frac{D_h(x,y)}{D_h(y,x)} \;\middle|\; (x,y) \in \operatorname{int} \operatorname{dom} h \times \operatorname{int} \operatorname{dom} h,\ x \neq y \right\} \in [0,1]. \tag{A.5}$$

Lemma 1. Given $h : X \to (-\infty, +\infty]$, D_h is general Bregman distance, and x, y, $z \in X$ such that $h(x)$, $h(y)$, $h(z)$ are finite and h is differentiable at y and z,

$$D_h(x,z) - D_h(x,y) - D_h(y,z) = \langle \nabla h(y) - \nabla h(z), x - y \rangle \tag{A.6}$$

Proof. The proof is straightforward as one can easily verify it by simply subtracting $D_h(y,z)$ and $D_h(x,y)$ from $D_h(x,z)$. □

A.3.2 Theorem 5.1 and Theorem 5.2

In this section, we first establish the convergence of Bregman proximal point algorithm, *i.e.*, **Theorem 5.1**, while our analysis is based on [158, 159, 160]. Further, we establish the convergence of inexact version Bregman proximal point algorithm, *i.e.*, **Theorem 5.2**, in which the subproblem in each iteration is computed inexactly within finite number of sub-iterations.

Note that here for simplicity we provide proof of $d(\mathbf{\Gamma}, \mathbf{\Gamma}^{(t)}) = D_h(\mathbf{\Gamma}, \mathbf{\Gamma}^{(t)})$, i.e., the IPOT case. We can analogously prove it for $d(\{\mathbf{\Gamma}_k\}, \{\mathbf{\Gamma}_k^{(t)}\}) = \sum_{k=1}^{K} \lambda_k D_h(\mathbf{\Gamma}_k, \mathbf{\Gamma}_k^{(t)})$, i.e.,

the IPOT-WB case, with very similar proof. This is because the latter is essentially just the weighted version of the former.

Before proving both theorems, we propose several fundamental lemmas. The first Lemma is the fundamental descent lemma, which is popularly used to analysis the convergence result of first-order methods.

Lemma 2. (Descent Lemma) Consider a closed proper convex function $f : X \to (-\infty, \infty]$ and for any $x \in X$ and $\beta^{(t)} > 0$, we have:

$$f(x^{(t+1)}) \leq f(x) + \beta^{(t)} \left[D_h(x, x^{(t)}) - D_h(x, x^{(t+1)}) - D_h(x^{(t+1)}, x^{(t)}) \right], \quad \forall x \in X. \tag{A.7}$$

Proof. The optimality condition of (Equation A.3) can be written as

$$\left(x - x^{(t+1)}\right)^T \left[\nabla f(x^{(t+1)}) + \beta^{(t)} \left(\nabla h(x^{(t+1)}) - \nabla h(x^{(t)}) \right) \right] \geq 0, \quad \forall x \in X.$$

Then with the convexity of f, we obtain

$$f(x) - f(x^{(t+1)}) + \beta^{(t)} \left(x - x^{(t+1)}\right)^T \left(\nabla h(x^{(t+1)}) - \nabla h(x^{(t)}) \right) \geq 0. \tag{A.8}$$

With (Equation A.6) it follows that

$$\left(x - x^{(t+1)}\right)^T \left(\nabla h(x^{(t+1)}) - \nabla h(x^{(t)}) \right) = D_h(x, x^{(t)}) - D_h(x, x^{(t+1)}) - D_h(x^{(t+1)}, x^{(t)}).$$

Substitute the above equation into (Equation A.8), we have

$$f(x^{(t+1)}) \leq f(x) + \beta^{(t)} \left[D_h(x, x^{(t)}) - D_h(x, x^{(t+1)}) - D_h(x^{(t+1)}, x^{(t)}) \right], \quad \forall x \in X.$$

□

Next, we prove the convergence result in **Theorem 5.1**.

Theorem 5.1 *Let $\{x^{(t)}\}$ be the sequence generated by the general Bregman proximal point algorithm with iteration (Equation A.3) where f is assumed to be continuous and convex. Further assume that $f^* = \min f(x) > -\infty$. Then we have that $\{f(x^{(t)})\}$ is non-increasing, and $f(x^{(t)}) \to f^*$. Further assume there exists η, s.t.*

$$f^* + \eta d(x) \leq f(x), \quad \forall x \in X, \tag{A.9}$$

The algorithm has linear convergence.

Proof. 1. First, we prove the sufficient decrease property:

$$f(x^{(t+1)}) \leq f(x^{(t)}) - \beta^{(t)}(1 + \alpha(h))D_h(x^{(t+1)}, x^{(t)}). \tag{A.10}$$

Let $x = x^{(t)}$ in (Equation A.7), we obtain

$$\begin{aligned} f(x^{(t+1)}) &\leq f(x^{(t)}) - \beta^{(t)} \left[D_h(x^{(t)}, x^{(t+1)}) + D_h(x^{(t+1)}, x^{(t)})\right] \\ &\leq f(x^{(t)}) - \beta^{(t)}(1 + \alpha(h))D_h(x^{(t+1)}, x^{(t)}). \end{aligned}$$

With the sufficient decrease property, it is obvious that $\{f(x^{(t)})\}$ is non-decreasing.

2. Summing (Equation A.10) from $i = 0$ to $i = t - 1$ and for simplicity assuming $\beta^{(t)} = \beta$, we have

$$\sum_{i=0}^{k-1} \left[\frac{1}{\beta^{(t)}} \left(f(x^{(i+1)}) - f(x^{(i)})\right)\right] \leq -\left[1 + \alpha(h)\right] \sum_{i=0}^{k-1} D_h(x^{(i+1)}, x^{(i)})$$

$$\Rightarrow \qquad \sum_{i=0}^{\infty} D_h(x^{(i+1)}, x^{(i)}) < \frac{1}{\beta(1 + \alpha(h))} f(x^{(0)}) < \infty,$$

which indicates that $D_h(x^{(i+1)}, x^{(i)}) \to 0$. Then summing (Equation A.7) from $i = 0$

to $i = t-1$, we have

$$k\left(f(x^{(t)}) - f(x)\right) \leq \sum_{i=0}^{t-1} \left(f(x^{(i+1)}) - f(x)\right) \leq \beta D_h(x, x^{(0)}) < \infty, \quad \forall x \in X.$$

Let $t \to \infty$, we have $\lim_{t\to\infty} f(x^{(t)}) \leq f(x)$ for every x, as a result we have $\lim_{k\to\infty} f(x^{(t)}) = f^*$.

3. Finally, we prove the convergence rate is linear. Assume $x^* = \operatorname{argmin}_x f(x)$ is the unique optimal solution. Denote $d(x) = D_h(x^*, x)$. Let also $\beta^{(t)} = \beta$, we will prove

$$\frac{d(x^{(t+1)})}{d(x^{(t)})} \leq \frac{1}{1+\frac{\eta}{\beta}} \tag{A.11}$$

Replace x with x^* in inequality (Equation A.7), we have

$$f(x^{(t+1)}) \leq f^* + \beta\left[d(x^{(t)}) - d(x^{(t+1)}) - D_h(x^{(t+1)}, x^{(t)})\right]. \tag{A.12}$$

Using assumption Equation A.9, we have

$$f^* + \eta d(x^{(t+1)}) \leq f(x^{(t+1)}) \tag{A.13}$$

Sum Equation A.12 and Equation A.13 up, we have

$$\begin{aligned}\frac{\eta}{\beta} d(x^{(t+1)}) &\leq d(x^{(t)}) - d(x^{(t+1)}) - D_h(x^{(t+1)}, x^{(t)}) \\ &\leq d(x^{(t)}) - d(x^{(t+1)})\end{aligned}$$

Therefore,

$$\frac{d(x^{(t+1)})}{d(x^{(t)})} \leq \frac{1}{1+\frac{\eta}{\beta}}$$

Therefore, we have a linear convergence in Bregman distance sense.

□

Assumption (Equation A.9) does not always hold when f is linear. In our specific case, x is bounded in $[0,1]^{m\times n}$. More rigorously, we can prove the following lemma.

Lemma 3. Assume $\mathcal{X}$ is a bounded polyhedron, x^* is unique, $d(x)$ is an arbitrary nonnegative convex function with $d(x*) = 0$. If f is linear, then there exist η, s.t. $f^* + \eta d(x) \leq f(x)$.

Proof. Since $\mathcal{X}$ is a bounded polyhedron, any $x \in \mathcal{X}$ can be expressed as $x = \sum_{i=0}^{n} \lambda_i e_i$, where e_i is the vertices of $\mathcal{X}$, n is finite, and $\sum \lambda_i = 1$. Also f is linear, so $f(x) = \sum_{i=0}^{n} \lambda_i f(e_i)$

Since f is linear, $\mathcal{X}$ is polyhedral and x^* is unique, x^* is a vertex of X. Denote $e_0 = x^*$.

Denote $\delta = \min_{i>0} f(e_i) - f^*$, then $\delta > 0$, or else x^* is not unique. Denote $d_{max} = \max_{i>0} d(e_i)$. Take

$$\eta = \delta / d_{max},$$

we have

$$\begin{aligned}
f^* + \eta d(x) &= f^* + \eta d(\sum_{i=0}^{n} \lambda_i e_i) \\
\text{(Jensen's Inequality)} &\leq f^* + \eta \sum_{i=0}^{n} \lambda_i d(e_i) \\
(d(e_0) = 0) \quad &\leq f^* + (1-\lambda_0)\eta d_{max} \\
&= f^* + (1-\lambda_0)\delta \\
&= \sum_{i=1}^{n} \lambda_i (f^* + \delta) + \lambda_0 f^* \\
&\leq \sum_{i=0}^{n} \lambda_i f(e_i) \\
&= f(x).
\end{aligned}$$

□

For more general cases, if x^* is not unique, we can divide the vertices as "optimal

vertices" and the rest vertices, instead of e_0 and the rest as above, the conclusion can be proved analogously. Furthermore, if $\mathcal{X}$ is not a polyhedron, as long as $\mathcal{X}$ is bounded, we can always prove the conclusion in a polyhedron $\mathcal{A}$ s.t. $\mathcal{X} \in \mathcal{A}$ and $x*$ is also the optimal solution of $\min_{x\in\mathcal{A}} f(x)$. Proof of more general cases can be found in [161] (This paper points out some fairly strong continuity properties that polyhedral multifunctions satisfy).

Inequality (Equation A.11) shows how the convergence rate is linked to β. This is the reason we claim in Section 4.1 that a smaller β would lead to quicker convergence in exact case.

From above, we showed that the general Bregman proximal point algorithm with constant step size can guarantee convergence to the optimal solution f^*, and has linear convergence rate with some assumptions. Further, we prove the convergence result for the general Bregman proximal point algorithm with inexact scheme in **Theorem 5.2**.

Theorem 5.2 *Let $\{x^{(t)}\}$ be the sequence generated by the general Bregman proximal point algorithm with inexact scheme (i.e., finite number of inner iterations are employed). Define an error sequence $\{e^{(t)}\}$ where*

$$e^{(t+1)} \in \beta^{(t)} \left[\nabla f(x^{(t+1)}) + \partial \iota_X(x^{(t+1)})\right] + \left[\nabla h(x^{(t+1)}) - \nabla h(x^{(t)})\right], \tag{A.14}$$

where ι_X is the indicator function of set X. If the sequence $\{e^{(t)}\}$ satisfies $\sum_{k=1}^{\infty} \|e^{(t)}\| < \infty$ and $\sum_{k=1}^{\infty} \langle e^{(t)}, x^{(t)}\rangle$ exists and is finite, then $\{x^{(t)}\}$ converges to x^∞ with $f(x^\infty) = f^*$. If the sequence $\{e^{(t)}\}$ satisfies that exist $\rho \in (0,1)$ such that$\|e^{(t)}\| \leq \rho^t$, $\langle e^{(t)}, x^{(t)}\rangle \leq \rho^t$ and with assumption (Equation A.9), then $\{x^{(t)}\}$ converges linearly.

Remark: If exact minimization is guaranteed in each iteration, the sequence $\{x^{(t)}\}$ will satisfy that

$$0 \in \beta^{(t)} \left[\nabla f(x^{(t+1)}) + \partial \iota_X(x^{(t+1)})\right] + \frac{1}{\beta^{(t)}} \left[\nabla h(x^{(t+1)}) - \nabla h(x^{(t)})\right].$$

As a result, with enough inner iteration, the guaranteed $e^{(t)}$ will goes to zero.

Proof. This theorem is extended from [159, Theorem 1], and we propose a brief proof here. The proof contains the following four steps:

1. We have for all $k \geq 0$, through the three point lemma

$$D_h(x, x^{(t+1)}) = D_h(x, x^{(t)}) - D_h(x^{(t+1)}, x^{(t)}) - \langle \nabla h(x^{(t)}) - \nabla h(x^{(t+1)}), x^{(t+1)} - x \rangle,$$

which indicates

$$D_h(x, x^{(t+1)}) = D_h(x, x^{(t)}) - D_h(x^{(t+1)}, x^{(t)}) - \langle \nabla h(x^{(t)}) - \nabla h(x^{(t+1)}) + e^{(t+1)}, x^{(t+1)} - x \rangle + \langle e^{(t+1)}, x$$

Since $\frac{1}{\beta^{(t)}} \left[e^{(t+1)} + \nabla h(x^{(t)}) - \nabla h(x^{(t+1)}) \right] \in \nabla f(x^{(t+1)}) + \partial \iota_X(x^{(t+1)})$ and $0 \in \nabla f(x^*) + \partial \iota_X(x^*)$ if x^* be the optimal solution, we have

$$\begin{aligned} & \langle \nabla h(x^{(t)}) - \nabla h(x^{(t+1)}) + e^{(t+1)}, x^{(t+1)} - x^* \rangle \\ = \quad & \beta^{(t)} \langle \left[\frac{1}{\beta^{(t)}} \left(\nabla h(x^{(t)}) - \nabla h(x^{(t+1)}) + e^{(t+1)} \right) \right] - 0, x^{(t+1)} - x^* \rangle \geq 0, \end{aligned}$$

because $\nabla f + \partial \iota_X$ is monotone ($f + \iota_X$ is convex). Further we have

$$D_h(x^*, x^{(t+1)}) \leq D_h(x^*, x^{(t)}) - D_h(x^{(t+1)}, x^{(t)}) + \langle e^{(t+1)}, x^{(t+1)} - x^* \rangle.$$

2. Summing the above inequality from $i = 0$ to $i = t - 1$, we have

$$D_h(x^*, x^{(t)}) \leq D_h(x^*, x^{(0)}) - \sum_{i=0}^{t-1} D_h(x^{(i+1)}, x^{(i)}) + \sum_{i=0}^{t-1} \langle e^{(i+1)}, x^{(i+1)} - x^* \rangle.$$

Since $\sum_{t=1}^{\infty} \|e^{(t)}\| < \infty$ and $\sum_{t=1}^{\infty} \langle e^{(t)}, x^{(t)} \rangle$ exists and is finite, we guarantee that

$$\bar{E}(x^*) = \sup_{t \geq 0} \left\{ \sum_{i=0}^{t-1} \langle e^{(i+1)}, x^{(i+1)} - x^* \rangle \right\} < \infty.$$

Together with $D_h(x^{(i+1)}, x^{(i)}) > 0$, we have

$$D_h(x^*, x^{(t)}) \leq D_h(x^*, x^{(0)}) + \bar{E}(x^*) < \infty,$$

which indicates

$$0 \leq \sum_{i=0}^{\infty} D_h(x^{(i+1)}, x^{(i)}) < D_h(x^*, x^{(0)}) + \bar{E}(x^*) < \infty,$$

and hence $D_h(x^{(i+1)}, x^{(i)}) \to 0$.

3. Based on the above two items, we know that the sequence $\{x^{(t)}\}$ must be bounded and has at least one limit point x^∞. The most delicate part of the proof is to establish that $0 \in \nabla f(x^\infty) + \partial \iota_X(x^\infty)$. Let $T = \nabla f + \partial \iota_X$, then T denotes the subdifferential mapping of a closed proper convex function $f + \iota_X$ (f is a linear function and X is a closed convex set). Let $\{t_j\}$ be the sub-sequence such that $x^{t_j} \to x^\infty$. Because $x^{t_j} \in X$ and X is a closed convex set, we know $x^\infty \in X$. We know that $D_h(x^*, x^{(t+1)}) \leq D_h(x^*, x^{(t)}) + \langle e^{(t+1)}, x^{(t+1)} - x^* \rangle$ and $\sum_{k=0}^{\infty} \langle e^{(t+1)}, x^{(t+1)} - x^* \rangle$ exists and is finite. From [162, Section 2.2], we guarantee that $\{D_h(x^*, x^{(t)})\}$ converges to $0 \leq d(x^*) < \infty$. Define $y^{(t+1)} := \lambda_k \left(\nabla h(x^{(t)}) - \nabla h(x^{(t+1)}) + e^{(t+1)} \right)$, we have

$$\lambda_k \langle y^{(t+1)}, x^{(t+1)} - x^* \rangle = D_h(x^*, x^{(t)}) - D_h(x^*, x^{(t+1)}) - D_h(x^{(t+1)}, x^{(t)}) + \langle e^{(t+1)}, x^{(t+1)} - x^* \rangle.$$

By taking the limit of both sides and $\lambda_k = \lambda > 0$, we obtain that

$$\langle y^{(t+1)}, x^{(t+1)} - x^* \rangle \to 0.$$

For the reason that y^{k_j+1} is a subgradient of $f + \iota_X$ at x^{k_j+1}, we have

$$f(x^*) \geq f(x^{k_j+1}) + \langle y^{k_j+1}, x^* - x^{k_j+1} \rangle, \quad x^* \in X, x^{k_j+1} \in X.$$

Further let $j \to \infty$ and using f is lower semicontinuous, $\langle y^{(t+1)}, x^{(t+1)} - x^* \rangle \to 0$, we obtain

$$f(x^*) \geq f(x^\infty), \quad x^\infty \in X$$

which implies that $0 \in \nabla f(x^\infty) + \iota_X(x^\infty)$.

4. Recall the inexact scheme (Equation A.14), we can equivalently guarantee that

$$(x - x^{(t+1)})^T \left\{ \beta^{(t)} \nabla f(x^{(t+1)}) + \left[\nabla h(x^{(t+1)}) - \nabla h(x^{(t)}) \right] - e^{(t+1)} \right\} \geq 0, \ \forall x \in X.$$

Together the convexity of f and the three point lemma, we obtain

$$f(x^{(t+1)}) \leq f(x) + \frac{1}{\beta^{(t)}} \left[D_h(x, x^{(t)}) - D_h(x, x^{(t+1)}) - D_h(x^{(t+1)}, x^{(t)}) - (x - x^{(t+1)})^T e^{(t+1)} \right].$$

Let $x = x^*$ in the above inequality and recall the assumption (Equation A.9), *i.e.*,

$$f(x) - f(x^*) \geq \eta d(x),$$

we have with $\beta^{(t)} = \beta$

$$\begin{aligned} \eta d(x^{(t+1)}) &\leq \frac{1}{\beta} \left[d(x^{(t)}) - d(x^{(t+1)}) \right] + \frac{1}{\beta} \left((x^{(t+1)} - x^*)^T e^{(t+1)} \right) \\ &\leq \frac{1}{\beta} \left[d(x^{(t)}) - d(x^{(t+1)}) \right] + \frac{1}{\beta} \left(C \| e^{(t+1)} \| + \langle x^{(t+1)}, e^{(t+1)} \rangle \right), \end{aligned}$$

where $C := \sup_{x \in X^*} \{\|x\|\}$. The second inequality is obtained through triangle inequality. Then

$$d^{(t+1)} \leq \mu d^{(t)} + \mu \left(C \| e^{(t+1)} \| + \langle x^{(t+1)}, e^{(t+1)} \rangle \right),$$

where $\mu = \frac{1}{1+\beta\eta} < 1$. With our assumptions and according to Theorem 2 and Corollary 2 in [163], we guarantee the generated sequence converges linearly in the

order of $\mathcal{O}\left(c^t\right)$, where $c = \sqrt{\frac{1+\max\{\mu,\rho\}}{2}} \in (0,1)$.

Based on the above four items, we guarantee the convergence results in this theorem.

□

APPENDIX B

APPENDIX ON SPOT

B.1 Network Architecture

B.1.1 No-sharing Network

The CNN architecture for experiments in Section subsection 3.6.3. Table Table B.1 shows the architecture of two mappings G_X and G_Y. The two mappings have identical architechture.

Table B.1: The CNN architecture for experiments of real datasets in Section subsection 3.6.3.

Input:	$z \in \mathbb{R}^{100} \sim \mathcal{N}(0, I)$	
	Convolution Filter	Activation
Deconv:	[4 × 4, 512, stride = 1, padding=0]	BN, ReLU
Deconv:	[4 × 4, 256, stride = 2, padding=1]	BN, ReLU
Deconv:	[4 × 4, 128, stride = 2, padding=1]	BN, ReLU
Deconv:	[4 × 4, 64, stride = 2, padding=1]	BN, ReLU
Deconv:	[4 × 4, 3, stride = 2, padding=1]	Tanh

Table Table B.2 shows the architecture of two discriminators λ_X, λ_Y. The two networks have identical architechture and do not share parameters.

B.1.2 Convolutional Network

The CNN architecture for USPS, MNIST and MNISTM. PReLU activation is applied [164]. Table Table B.3 shows the architecture of two generators G_X and G_Y. The last column in Table Table B.3 means whether G_X and G_Y share the same parameter.

Table Table B.4 shows the architecture of two discriminators λ_X, λ_Y, and two classifiers D_X, D_Y. The last column in Table Table B.3 uses $(\cdot, \cdot)$ to denote which group of

Table B.2: The CNN architecture of λ_X, λ_Y for experiments of real datasets in Section subsection 3.6.3.

Input:	Image $x \in \mathbb{R}^{64\times64\times3} \sim \mu$ or ν	
	Convolution Filter	Activation
Conv:	[4 × 4, 64, stride = 1, padding=0]	ReLU
Conv:	[4 × 4, 128, stride = 2, padding=1]	BN, ReLU
Conv:	[4 × 4, 256, stride = 2, padding=1]	BN, ReLU
Conv:	[4 × 4, 512, stride = 2, padding=1]	BN, ReLU
Conv:	[4 × 4, 1, stride = 1, padding=0]	−

Table B.3: The CNN generater architecture for USPS, MNIST and MNISTM. $ch = 1$ for USPS and MNIST; $ch = 3$ for MNISTM.

Input:	$z \in \mathbb{R}^{100} \sim \mathcal{N}(0, I)$		
	Convolution Filter	Activation	Shared
Deconv:	[4 × 4, 1024, stride = 1, padding=0]	BN, PReLU	True
Deconv:	[3 × 3, 512, stride = 2, padding=1]	BN, PReLU	True
Deconv:	[3 × 3, 256, stride = 2, padding=1]	BN, PReLU	True
Deconv:	[3 × 3, 128, stride = 2, padding=1]	BN, PReLU	True
Deconv:	[3 × 6, ch, stride = 1, padding=1]	Sigmoid	False

discriminators share the same parameter.

Table B.4: The CNN discriminator architecture for USPS, MNIST and MNISTM. $ch = 1$ for USPS and MNIST; $ch = 3$ for MNISTM. $ch_o = 1$ for λ_X and λ_Y; $ch_o = 10$ for D_X and D_Y.

Input:	Image $x \in \mathbb{R}^{28\times28\times ch} \sim \mu$ or ν		
	Convolution Filter	Activation	Shared
Conv:	[5 × 5, 20, stride = 1, padding=0]	MaxPooling(2,2)	(λ_X, D_X);(λ_Y, D_Y)
Conv:	[5 × 5, 50, stride = 1, padding=0]	MaxPooling(2,2)	$(\lambda_X, \lambda_Y, D_X, D_Y)$
Conv:	[4 × 4, 500, stride = 1, padding=0]	PReLU	$(\lambda_X, \lambda_Y, D_X, D_Y)$
Conv:	[1 × 1, ch_o, stride = 1, padding=0]	−	$(\lambda_X); (\lambda_Y); (D_X, D_Y)$

B.1.3 Residual Network

The ResNet architecture for SVHN $\rightarrow$ MNIST. Table Table B.5 shows the architecture of two generators G_X and G_Y. The last column in Table Table B.5 means whether G_X and G_Y share the same parameter. The Residual block is the same as the one in [56].

Table B.5: The ResNet generater architecture for SVHN $\rightarrow$ MNIST. $ch = 1$ for MNIST; $ch = 3$ for SVHN.

Input:	$z \in \mathbb{R}^{100} \sim \mathcal{N}(0, I)$		
	Layer Size	Activation	Shared
Linear:	$100 \rightarrow 4 \times 4 \times 128$	–	True
ResBlocks:	[128, Up-sampling]	–	True
ResBlocks:	[128, Up-sampling]	–	True
ResBlocks:	[128, Up-sampling]	BN,PReLU	True
Conv:	[3×3, ch, stride = 1, padding =0]	Sigmoid	False

Table Table B.6 shows the architecture of two discriminators λ_X, λ_Y, and two classifiers D_X, D_Y. The last column in Table Table B.6 uses $(\cdot, \cdot)$ to denote which group of discriminators share the same parameter.

Table B.6: The ResNet discriminator architecture for SVHN $\rightarrow$ MNIST. $ch = 1$ for MNIST; $ch = 3$ for SVHN. $ch_o = 1$ for λ_X and λ_Y; $ch_o = 10$ for D_X and D_Y.

Input:	Image $x \in \mathbb{R}^{28 \times 28 \times ch} \sim \mu$ or ν		
	Layer Size	Activation	Shared
ResBlocks:	[128, Down-Sampling]	–	(λ_X, D_X);(λ_Y, D_Y)
ResBlocks:	[128, Down-Sampling]	–	$(\lambda_X, \lambda_Y, D_X, D_Y)$
ResBlocks:	[128, Down-Sampling]	–	$(\lambda_X, \lambda_Y, D_X, D_Y)$
Conv:	[4×4, 500, stride = 1, padding=0]	PReLU	$(\lambda_X, \lambda_Y, D_X, D_Y)$
Conv:	[1×1, ch_o, stride = 1, padding=0]	–	(λ_X); (λ_Y); (D_X, D_Y)

APPENDIX C

APPENDIX ON SOFT

C.1 Theoretical Guarantees

First, we show that after adding entropy regularization the problem is differentiable.

Theorem 1. For any $\epsilon > 0$, SOFT top-k operator: $\mathcal{X} \mapsto A^\epsilon$ is differentiable, as long as the cost C_{ij} is differentiable with respect to x_i for any i, j. Moreover, the Jacobian matrix of SOFT top-k operator always has a nonzero entry for any $\mathcal{X} \in \mathbb{R}^n$.

Proof. We first prove the differentiability. This part of proof mirrors the proof in [89]. By Sinkhorn's scaling theorem,

$$\Gamma^{*,\epsilon} = \mathrm{diag}(e^{\frac{\xi^*}{\epsilon}}) e^{-\frac{C}{\epsilon}} \mathrm{diag}(e^{\frac{\zeta^*}{\epsilon}}).$$

Therefore, since C_{ij} is differentiable, $\Gamma^{*,\epsilon}$ is differentiable if (ξ^*, ζ^*) is differentiable as a function of input scores X.

Let us set

$$\mathcal{L}(\xi, \zeta; \mu, \nu, C) = \xi^T \mu + \zeta^T \nu - \epsilon \sum_{i,j=1}^{n,m} e^{-\frac{C_{ij} - \xi_i - \zeta_j}{\epsilon}}.$$

and recall that $(\xi^*, \zeta^*) = \mathrm{argmax}_{\xi,\zeta} L(\xi, \zeta; \mu, \nu, C)$. The differentiability of (ξ^*, ζ^*) is proved using the Implicit Function theorem and follows from the differentiability and strong convexity in (ξ^*, ζ^*) of the function $\mathcal{L}$.

Now we prove that dA^ϵ/dx_ℓ always has a nonzero entry for $l = 1, \cdots, n$. First, we prove that for any $\ell \in \{1, \cdots, n\}$, $d\Gamma^{*,\epsilon}/dx_\ell$ always has a nonzero entry. We will prove it

by contradiction. Specifically, the KKT conditions for the stationarity are as follows

$$\xi_i^* + \zeta_j^* = (x_i - y_j)^2 - \epsilon \log \Gamma_{ij}^{*,\epsilon}, \quad \forall i = 1, \cdots, n, j = 1, \cdots, m.$$

If we view the above formula as a linear equation set of the dual variables, it has nm equations and $m + n$ variables. Therefore, there are $nm - m - n$ redundant equations. Suppose one of the scores x_ℓ, has an infinitesimal change δx_ℓ. Assuming $\Gamma^{*,\epsilon}$ does not change, we have a new set of linear equations,

$$\begin{aligned}
\xi_i^* + \zeta_j^* &= (x_i - y_j)^2 - \epsilon \log \Gamma_{ij}^{*,\epsilon}, \quad \forall i \neq \ell, \\
\xi_\ell^* + \zeta_j^* &= (x_\ell + \delta x_\ell - y_j)^2 + \delta C_{\ell j} - \epsilon \log \Gamma_{\ell j}^{*,\epsilon}.
\end{aligned}$$

Easy to verify that this set of linear equations has no solution. Therefore, there must be at least one entry in $\Gamma^{*,\epsilon}$ has changed. As a result, $d\Gamma^{*,\epsilon}/dx_\ell$ always has a nonzero entry. We denote this entry as $\Gamma_{i'j'}^{*,\epsilon}$. Since $\Gamma_{i'j'}^{*,\epsilon} + \Gamma_{i',3-j'}^{*,\epsilon} = \mu_{i'}$, we have

$$\frac{d\Gamma_{i',3-j'}^{*,\epsilon}}{dx_\ell} = -\frac{d\Gamma_{i'j'}^{*,\epsilon}}{dx_\ell} \neq 0.$$

Therefore, there must be a nonzero entry in the first column of $d\Gamma^{*,\epsilon}/dx_\ell$. Recall A^ϵ is the first column of $\Gamma^{*,\epsilon}$. As a result, there must be a nonzero entry in dA^ϵ/dx_ℓ for any $\ell \in \{1, \cdots, n\}$.

□

Second, we would like to know after smoothness relaxation, how much bias is introduced to A^ϵ.

Lemma 4. Denote the feasible set of optimal transport problem as $\Delta = \{\Gamma : \Gamma \in [0,1]^{n \times m}, \Gamma \mathbf{1}_m = \mu, \Gamma \mathbf{1}_n = \nu\}$. Assume the optimal transport plan is unique. Denote

Γ^* as the optimal transport plan,

$$\Gamma^* = \underset{\Gamma \in \Delta}{\operatorname{argmin}} f(\Gamma) = \underset{\Gamma \in \Delta}{\operatorname{argmin}} \langle C, \Gamma \rangle,$$

and $\Gamma^{*,\epsilon}$ as the entropy regularized transport plan,

$$\Gamma^{*,\epsilon} = \underset{\Gamma \in \Delta}{\operatorname{argmin}} f^\epsilon(\Gamma) = \underset{\Gamma \in \Delta}{\operatorname{argmin}} f(\Gamma) - \epsilon H(\Gamma) = \underset{\Gamma \in \Delta}{\operatorname{argmin}} \langle C, \Gamma \rangle + \epsilon \sum_{i,j} \Gamma_{ij} \ln \Gamma_{ij}.$$

We can bound the difference between Γ^* and $\Gamma^{*,\epsilon}$ to be

$$\|\Gamma^* - \Gamma^{*,\epsilon}\|_F \leq \epsilon \frac{(\ln n + \ln m)}{B},$$

where $\| \cdot \|_F$ is the Frobenius norm, and B is a positive constant irrelevant to ϵ.

Proof. Note that $H(\Gamma)$ is the entropy function. Since $0 \leq \Gamma_{ij} \leq 1$ and $\sum_{ij} \Gamma_{ij} = 1$ for any $\Gamma \in \Delta$, we can view Δ as the subset of a simplex. Therefore,

1. $H(\Gamma)$ is non-negative.
2. The maximum of $H(\Gamma)$ in the simplex can be obtained at $\Gamma_{ij} \equiv \frac{1}{nm}$. Therefore the maximum value is $(\ln n + \ln m)$.

Therefore, $0 \leq H(\Gamma) \leq (\ln n + \ln m)$ for any $\Gamma \in \Delta$.

Since $H(\Gamma) \geq 0$, we have $f^\epsilon(\Gamma) \leq f(\Gamma)$ for any $\Gamma \in \Delta$. As a result, we have $f^\epsilon(\Gamma^{*,\epsilon}) \leq f(\Gamma^*)$. In other words, we have

$$\langle C, \Gamma^{*,\epsilon} \rangle - \epsilon H(\Gamma^{*,\epsilon}) - \langle C, \Gamma^* \rangle \leq 0.$$

Therefore,

$$\langle C, \Gamma^{*,\epsilon} - \Gamma^* \rangle = \langle C, \Gamma^{*,\epsilon} \rangle - \langle C, \Gamma^* \rangle \leq \epsilon H(\Gamma^{*,\epsilon}) \leq \epsilon(\ln n + \ln m).$$

Since the optimal transport problem is a linear optimization problem, Γ^* is one of the vertices of Δ. Denote $e_0, e_1, \cdots, e_J$ as the vertices of Δ, and without loss of generality we assume $e_0 = \Gamma^*$. Since $\Gamma^{*,\epsilon} \in \Delta$, we can denote $\Gamma^{*,\epsilon} = \sum_{j=0}^{J} \lambda_j e_j$, where $\lambda_j \geq 0$, and $\sum_j \lambda_j = 1$. Since Γ^* is unique, we have

$$\langle C, e_j - e_0 \rangle > 0, \quad \forall j = 1, \cdots, J.$$

Denote $B_j = \langle C, e_j - e_0 \rangle$. Since the space we are considering is Euclidean space (if we reshape the matrices into vectors), we can write the inner product as

$$B_j = \langle C, e_j - e_0 \rangle = \|C\|_F \|e_j - e_0\|_F \cos\theta_{(C, e_j - e_0)} > 0.$$

So we have $\cos\theta_{(C,e_j-e_0)} > 0$. In other words, the angle between C and $e_j - e_0$ is always smaller than $\frac{\pi}{2}$. Therefore, the angle between C and the affine combination of $e_j - e_0$, namely $\sum_{j=0}^{J} \lambda_j (e_j - e_0)$, is also smaller than $\frac{\pi}{2}$. More specifically, we have

$$\cos\theta_{(C,\Gamma^{*,\epsilon}-\Gamma^*)} = \cos\theta_{(C,\sum_{j=0}^{J}\lambda_j(e_j-e_0))} \geq \min_j \cos\theta_{(C,e_j-e_0)} = \min_j \frac{B_j}{\|C\|_F \|e_j - e_0\|_F}.$$

Therefore, we have

$$\|\Gamma^{*,\epsilon} - \Gamma^*\|_F = \frac{\langle C, \Gamma^{*,\epsilon} - \Gamma^* \rangle}{\|C\|_F \cos\theta_{(C,\Gamma^{*,\epsilon}-\Gamma^*)}} \leq \frac{\epsilon(\ln n + \ln m)}{\|C\|_F \min_j \frac{B_j}{\|C\|_F \|e_j - e_0\|_F}} = \frac{\epsilon(\ln n + \ln m)}{\min_j \frac{B_j}{\|e_j - e_0\|_F}}.$$

Denote $B = \min_j \frac{B_j}{\|e_j - e_0\|_F}$, and we have the conclusion. □

Remark 4. In Theorem 1 we restricted the optimal solution to be unique, only for clarity purpose. If it is not unique, similar conclusion holds, except that the proof is more tedious – instead of divide the vertices into e_0 and others, we need to divide it into the vertices that are optimal solutions and the others.

Lemma 5. At each of the vertices of Δ, the entries of Γ are either 0 or $1/n$ for $\Gamma \in \Delta$.

Proof. The key idea is to prove by contradiction: If there exist i, j such that $\Gamma_{ij} \in (0, 1/n)$, then Γ cannot be a vertex.

To ease the discussion, we denote $Z = n\Gamma$. We will first prove that the entries of Z are either 0 or 1 at the vertices.

Notice that

$$\begin{aligned} &Z_{i,1} + Z_{i,2} = 1, \quad \forall i = 1, \cdots, n, \\ &\sum_i Z_{i,1} = k, \\ &\sum_i Z_{i,2} = n - k. \end{aligned}$$

If there exists an entry $Z_{i',j'} \in (0, 1)$, then

1. $Z_{i',3-j'} \in (0, 1)$.
2. there must exist $i'' \neq i'$, such that $Z_{i'',j'} \in (0, 1)$. This is because $\sum_{i=1}^{n} Z_{i,j}$ is an integer, and $Z_{i',j'}$ is not.
3. As a result, $Z_{i'',3-j'} \in (0, 1)$.

Therefore, consider $\delta \in (-\min\{1 - Z_{i'.j'}, Z_{i',j'}\}, \min\{1 - Z_{i'.j'}, Z_{i',j'}\})$ and denote

$$\widetilde{Z}^{(1)}_{ij} = \begin{cases} Z_{i',j'} + \delta, \text{ if } i = i', j = j', \\ Z_{i',3-j'} - \delta, \text{ if } i = i', j = 3 - j', \\ Z_{i'',j'} - \delta, \text{ if } i = i'', j = j', \\ Z_{i'',3-j'} + \delta, \text{ if } i = i'', j = 3 - j', \\ Z_{i,j}, \text{ otherwise.} \end{cases}$$

$$\widetilde{Z}^{(2)}_{ij} = \begin{cases} Z_{i',j'} - \delta, \text{ if } i = i', j = j', \\ Z_{i',3-j'} + \delta, \text{ if } i = i', j = 3 - j', \\ Z_{i'',j'} + \delta, \text{ if } i = i'', j = j', \\ Z_{i'',3-j'} - \delta, \text{ if } i = i'', j = 3 - j', \\ Z_{i,j}, \text{ otherwise.} \end{cases}$$

We can easily verify that $\widetilde{Z}^{(1)}/n, \widetilde{Z}^{(2)}/n \in \Delta$, and also $Z = (\widetilde{Z}^{(1)} + \widetilde{Z}^{(2)})/2$. Therefore, Z cannot be a vertex.

□

Lemma 6. Given a set of scalar $\{x_1, \cdots, x_n\}$, we sort it to be $\{x_{\sigma_1}, \cdots, x_{\sigma_n}\}$. If Euclidean square cost is adopted, Γ^* has the following form,

$$\Gamma^*_{ij} = \begin{cases} 1/n, \text{ if } i = \sigma_\ell, j = 1, \ell \leq k \\ 0, \text{ if } i = \sigma_\ell, j = 1, k < \ell \leq n \\ 1/n, \text{ if } i = \sigma_\ell, j = 2, k < \ell \leq n \\ 0, \text{ if } i = \sigma_\ell, j = 2, \ell \leq k \end{cases}$$

And $\min_j \frac{B_j}{\|e_j - e_0\|_F}$ is attained at at a vertex Γ^{**}, where $\Gamma^{**}_{ij} = \Gamma^*_{ij}$ except that the σ_k-th row

and the σ_{k+1}-th row are swapped. As a result, we have

$$\min_j \frac{B_j}{\|e_j - e_0\|_F} = n(x_{\sigma_{k+1}} - x_{\sigma_k}).$$

Proof. From Lemma 5, in each vertex the entries of Γ is either 0 or $1/n$. Also, $\Gamma^* \in \Delta = \{\Gamma : \Gamma \in [0,1]^{n\times m}, \Gamma \mathbf{1}_m = \mathbf{1}_n/n, \Gamma \mathbf{1}_n = [k/n, (n-k)/n]^\top\}$. Therefore, for the j-th vertex, there are k entries with value $1/n$ in the first row of Γ. Denote the row indices of these k entries as $\mathcal{I}_j$, and $\Omega = \{1, \cdots, n\}$. Then for each vertex we have

$$\begin{aligned}
&\Gamma_{i,1} = 1/n, \quad \forall i \in \mathcal{I}_j \\
&\Gamma_{i,1} = 0, \quad \forall i \in \Omega \backslash \mathcal{I}_j \\
&\Gamma_{i,2} = 1/n, \quad \forall i \in \Omega \backslash \mathcal{I}_j \\
&\Gamma_{i,2} = 0, \quad \forall i \in \mathcal{I}_j.
\end{aligned}$$

Denote $\mathcal{I}^* = \{\sigma_1, \cdots, \sigma_k\}$. We now prove that $\mathcal{I}^*$ corresponds to the optimal solution Γ^*. This is because for any $j \in \{1, \cdots, J\}$

$$\begin{aligned}
\Gamma(\mathcal{I}_j) - \Gamma(\mathcal{I}^*) &= \left(\sum_{i\in\mathcal{I}_j} x_i^2 + \sum_{i\in\Omega\backslash\mathcal{I}_j} (x_i - 1)^2\right) - \left(\sum_{i\in\mathcal{I}^*} x_i^2 + \sum_{i\in\Omega\backslash\mathcal{I}^*} (x_i - 1)^2\right) \\
&= \left(\sum_{i\in\Omega} x_i^2 - \sum_{i\in\Omega\backslash\mathcal{I}_j} 2x_i + (n-k)\right) - \left(\sum_{i\in\Omega} x_i^2 - \sum_{i\in\Omega\backslash\mathcal{I}^*} 2x_i + (n-k)\right) \\
&= 2\left(\sum_{i\in\Omega\backslash\mathcal{I}^*} x_i - \sum_{i\in\Omega\backslash\mathcal{I}_j} x_i\right) \geq 0,
\end{aligned}$$

where the last step is because the elements with indices $\Omega\backslash\mathcal{I}_j$ is the largest $n-k$ elements. Therefore we have $\Gamma(\mathcal{I}^*) = \Gamma^*$.

Now let's compute $\min_{j\neq 0} B_j/\|e_j - e_0\|$. Denote set subtraction $\mathcal{A} - \mathcal{B}$ as the set if

elements that belongs to $\mathcal{A}$ but do not belong to $\mathcal{B}$, and $|\mathcal{A}|$ as the number of elements in $\mathcal{A}$.

$$\begin{aligned}\frac{B_j}{\|e_j - e_0\|} &= \frac{B_j}{\|\Gamma(\mathcal{I}_j) - \Gamma(\mathcal{I}^*)\|} \\ &= 2\frac{\sum_{i\in\Omega\backslash\mathcal{I}^*} x_i - \sum_{i\in\Omega\backslash\mathcal{I}_j} x_i}{2\sqrt{|\mathcal{I}^* - \mathcal{I}_j|}/n} \\ &= n\frac{\sum_{i\in(\mathcal{I}_j - \mathcal{I}^*)} x_i - \sum_{i\in(\mathcal{I}^* - \mathcal{I}_j)} x_i}{\sqrt{|\mathcal{I}^* - \mathcal{I}_j|}},\end{aligned}$$

where the second line can be obtained by substituting the definition of B_j. Notice that $\mathcal{I}_j - \mathcal{I}^* \in \Omega\backslash\mathcal{I}^*$ and $\mathcal{I}^* - \mathcal{I}_j \in \mathcal{I}^*$. Any element with index in $\Omega\backslash\mathcal{I}^*$ is larger than any element in $\mathcal{I}^*$ by at least $x_{\sigma_{K+1}} - x_{\sigma_K}$. Then we have

$$\begin{aligned}\frac{B_j}{\|e_j - e_0\|} &= N\frac{\sum_{i\in(\mathcal{I}_j - \mathcal{I}^*)} x_i - \sum_{i\in(\mathcal{I}^* - \mathcal{I}_j)} x_i}{\sqrt{|\mathcal{I}^* - \mathcal{I}_j|}} \\ &\geq N\frac{|\mathcal{I}^* - \mathcal{I}_j|(x_{\sigma_{K+1}} - x_{\sigma_K})}{\sqrt{|\mathcal{I}^* - \mathcal{I}_j|}} \\ &\geq N(x_{\sigma_{K+1}} - x_{\sigma_K}),\end{aligned}$$

where the last step is because for $j \neq 0$, $|\mathcal{I}^* - \mathcal{I}_j|$ is at least 1.

Also notice that the value $n(x_{\sigma_{k+1}} - x_{\sigma_k})$ can be attained at $\mathcal{I}_{j^*} = \{\sigma_1, \cdots, \sigma_{k-1}, \sigma_{k+1}\}$. Therefore we have

$$\min_j \frac{B_j}{\|e_j - e_0\|} = n(x_{\sigma_{k+1}} - x_{\sigma_k}).$$

□

Theorem 2. Given a distinct sequence $\mathcal{X}$ and its sorting permutation σ, with Euclidean square cost function, for the proposed top-k solver we have

$$\|\Gamma^{*,\epsilon} - \Gamma^*\| \leq \frac{\epsilon(\ln n + \ln 2)}{n(x_{\sigma_{k+1}} - x_{\sigma_k})}.$$

Proof. This is a direct conclusion with Lemma 4 and Lemma 6. □

C.2 The Expression of the Gradient of A^ϵ

In this section we will derive the expression of dA^ϵ/dx_i. We first list a few reminders that will be used later:

- $\{x_i\}_{i=1}^n$ is a scalar set to be solved for top-k. $\{y_j\}_{j=1}^m$ is taken to be $\{0, 1\}$.
- $C \in \mathbb{R}^{n\times m}$ is the cost matrix, usually defined as $C_{ij} = (x_i - y_j)^2$.
- The loss function of entropic optimal transport is

$$\Gamma^{*,\epsilon} = \underset{\Gamma\in\Delta}{\operatorname{argmin}} f^\epsilon(\Gamma) = \underset{\Gamma\in\Delta}{\operatorname{argmin}} \langle C, \Gamma\rangle + \epsilon \sum_{i,j} \Gamma_{ij} \ln \Gamma_{ij},$$

  where $\Delta = \{\Gamma : \Gamma \in [0, 1]^{n\times m}, \Gamma\mathbf{1}_m = \mu, \Gamma\mathbf{1}_n = \nu\}$.

- The dual problem of the above optimization problem is

$$\xi^*, \zeta^* = \underset{\xi,\zeta}{\operatorname{argmax}} \mathcal{L}(\xi, \zeta; C),$$

  where

$$\mathcal{L}(\xi, \zeta; C) = \xi^\top \mu + \zeta^\top \nu - \epsilon \sum_{i,j=1}^{n,m} e^{-\frac{C_{ij}-\xi_i-\zeta_j}{\epsilon}}.$$

  And it is connected to the prime form by

$$\Gamma^{*,\epsilon} = \operatorname{diag}(e^{\frac{\xi^*}{\epsilon}}) e^{-\frac{C}{\epsilon}} \operatorname{diag}(e^{\frac{\zeta^*}{\epsilon}}).$$

  The converged p, q in Algorithm Algorithm 4 is actually $e^{\frac{\xi^*}{\epsilon}}$ and $e^{\frac{\zeta^*}{\epsilon}}$.

If we obtain the expression for $\frac{d\xi^*}{dC}$ and $\frac{d\zeta^*}{dC}$, we can obtain the expression for $\frac{dA^\epsilon}{dx_i}$.

In this section only, we denote $\Gamma = \Gamma^{*,\epsilon}$, to shorten the notation. The multiplication of 3rd-order tensors mirrors the multiplication of matrices: we always use the last dimension of the first input to multiplies the first dimension of the second input. We denote $\bar{b} = b_{:-1}$ as b removing the last entry, $\bar{\nu} = \nu_{:-1}$ as ν removing the last entry, $\bar{\Gamma} = \Gamma_{:,:-1}$ as Γ removing the last column.

Theorem 8. $\frac{d\xi^*}{dC}$ and $\frac{d\zeta^*}{dC}$ have the following expression,

$$\begin{bmatrix} \frac{d\xi^*}{dC} \\ \frac{d\zeta^*}{dC} \end{bmatrix} = \begin{bmatrix} -H^{-1}D \\ \mathbf{0} \end{bmatrix}$$

where $-H^{-1}D \in \mathbb{R}^{(n+m-1)\times n\times m}$, $\mathbf{0} \in \mathbb{R}^{1\times n\times m}$, and

$$D_{\ell ij} = \frac{1}{\epsilon}\begin{cases} \delta_{\ell i}\Gamma_{ij}, \ell = 1, \cdots, n \\ \delta_{\ell j}\Gamma_{ij}, \ell = n+1, \cdots, n+m-1 \end{cases}$$

$$H^{-1} = -\epsilon \begin{bmatrix} (\mathrm{diag}(\mu))^{-1} + (\mathrm{diag}(\mu))^{-1}\bar{\Gamma}\mathcal{K}^{-1}\bar{\Gamma}^T(\mathrm{diag}(\mu))^{-1} & -(\mathrm{diag}(\mu))^{-1}\bar{\Gamma}\mathcal{K}^{-1} \\ -\mathcal{K}^{-1}\bar{\Gamma}^T(\mathrm{diag}(\mu))^{-1} & \mathcal{K}^{-1} \end{bmatrix}$$

$$\mathcal{K} = \mathrm{diag}(\bar{\nu}) - \bar{\Gamma}^T(\mathrm{diag}(\mu))^{-1}\bar{\Gamma}.$$

Proof. Notice that there is one redundant dual variable, since $\mu\mathbf{1}_N = \nu\mathbf{1}_M = 1$. Therefore, we can rewrite $\mathcal{L}(\xi, \zeta; C)$ as

$$\mathcal{L}(\xi, \bar{\zeta}; C) = \xi^T\mu + \bar{\zeta}^T\bar{\nu} - \epsilon\sum_{i,j=1}^{n,m-1} e^{\frac{-C_{ij}+\xi_i+\zeta_j}{\epsilon}} - \epsilon\sum_{i=1}^{n} e^{\frac{-C_{im}+\xi_i}{\epsilon}}.$$

Denote

$$\phi(\xi, \bar{\zeta}, C) = \frac{d\mathcal{L}(\xi, \bar{\zeta}; C)}{d\xi} = \mu - F\mathbf{1}_m, \tag{C.1}$$

$$\psi(\xi, \bar{\zeta}, C) = \frac{d\mathcal{L}(\xi, \bar{\zeta}; C)}{d\bar{\zeta}} = \bar{\nu} - \bar{F}^\top\mathbf{1}_n, \tag{C.2}$$

where

$$
\begin{aligned}
&F_{ij} = e^{\frac{-C_{ij}+\xi_i+\zeta_j}{\epsilon}}, \quad \forall i = 1, \cdots, n, \quad j = 1, \cdots, m-1 \\
&F_{im} = e^{\frac{-C_{im}+\xi_i}{\epsilon}}, \quad \forall i = 1, \cdots, n, \\
&\bar{F} = F_{:,:-1}.
\end{aligned}
$$

Since $(\xi^*, \bar{\zeta}^*)$ is a maximum of $\mathcal{L}(\xi, \bar{\zeta}; C)$, we have

$$
\begin{aligned}
\phi(\xi^*, \bar{\zeta}^*, C) = 0, \\
\psi(\xi^*, \bar{\zeta}^*, C) = 0.
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
\frac{d\phi(\xi^*, \bar{\zeta}^*, C)}{dC} &= \frac{\partial \phi(\xi^*, \bar{\zeta}^*, C)}{\partial C} + \frac{\partial \phi(\xi^*, \bar{\zeta}^*, C)}{\partial \xi^*}\frac{d\xi^*}{dC} + \frac{\partial \phi(\xi^*, \bar{\zeta}^*, \mu, \nu, C)}{\partial \bar{\zeta}^*}\frac{d\bar{\zeta}^*}{dC} = 0, \\
\frac{d\psi(\xi^*, \bar{\zeta}^*, C)}{dC} &= \frac{\partial \psi(\xi^*, \bar{\zeta}^*, C)}{\partial C} + \frac{\partial \psi(\xi^*, \bar{\zeta}^*, C)}{\partial \xi^*}\frac{d\xi^*}{dC} + \frac{\partial \psi(\xi^*, \bar{\zeta}^*, C)}{\partial \bar{\zeta}^*}\frac{d\bar{\zeta}^*}{dC} = 0.
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
\begin{bmatrix} \frac{d\xi^*}{dC} \\ \frac{d\bar{\zeta}^*}{dC} \end{bmatrix} &= - \begin{bmatrix} \frac{\partial \phi(\xi^*, \bar{\zeta}^*, C)}{\partial \xi^*} & \frac{\partial \phi(\xi^*, \bar{\zeta}^*, C)}{\partial \bar{\zeta}^*} \\ \frac{\partial \psi(\xi^*, \bar{\zeta}^*, C)}{\partial \xi^*} & \frac{\partial \psi(\xi^*, \bar{\zeta}^*, C)}{\partial \bar{\zeta}^*} \end{bmatrix}^{-1} \begin{bmatrix} \frac{\partial \phi(\xi^*, \bar{\zeta}^*, C)}{\partial C} \\ \frac{\partial \psi(\xi^*, \bar{\zeta}^*, C)}{\partial C} \end{bmatrix} \\
&\triangleq -H^{-1} \begin{bmatrix} D^{(1)} \\ D^{(2)} \end{bmatrix} \\
&\triangleq -H^{-1} D.
\end{aligned}
$$

Now let's compute each of the terms.

$$\frac{\partial \phi(\xi^*, \bar{\zeta}^*, C)_h}{\partial C_{ij}} = -\frac{\partial [F\mathbf{1}_m]_h}{\partial C_{ij}} = -\frac{\partial}{\partial C_{ij}} \left(\sum_{\ell=1}^{m-1} e^{\frac{-C_{h\ell}+a_h+b_\ell}{\epsilon}} + e^{\frac{-C_{hm}+a_h}{\epsilon}} \right)$$
$$= \frac{1}{\epsilon}\delta_{hi} F_{ij} = \frac{1}{\epsilon}\delta_{hi}\Gamma_{ij}$$
$$\forall h = 1, \cdots, n, \quad i = 1, \cdots, n, \quad j = 1, \cdots, m$$
$$\frac{\partial \psi(\xi^*, \bar{\zeta}^*, C)_\ell}{\partial C_{ij}} = -\frac{\partial [\bar{F}^\top \mathbf{1}_n]_\ell}{\partial C_{ij}} = -\frac{\partial}{\partial C_{ij}} \sum_{h=1}^{n} e^{\frac{-C_{h\ell}+a_h+b_\ell}{\epsilon}}$$
$$= \frac{1}{\epsilon}\delta_{\ell j} F_{ij} = \frac{1}{\epsilon}\delta_{\ell j}\Gamma_{ij}$$
$$\forall \ell = 1, \cdots, m-1, \quad i = 1, \cdots, n, \quad j = 1, \cdots, m$$
$$\frac{\partial \phi(\xi^*, \bar{\zeta}^*, C)_h}{\partial \xi_i^*} = -\frac{\partial [F\mathbf{1}_m]_h}{\partial \xi_i^*} = -\frac{\partial}{\partial \xi_i^*} \left(\sum_{\ell=1}^{m-1} e^{\frac{-C_{h\ell}+a_h+b_\ell}{\epsilon}} + e^{\frac{-C_{hm}+a_h}{\epsilon}} \right)$$
$$= -\frac{1}{\epsilon}\delta_{hi} \sum_{\ell=1}^{m} F_{h\ell} = -\frac{1}{\epsilon}\delta_{hi}\mu_h$$
$$\forall h = 1, \cdots, n, \quad i = 1, \cdots, n$$
$$\frac{\partial \phi(\xi^*, \bar{\zeta}^*, C)_h}{\partial \bar{\zeta}_j^*} = -\frac{\partial [F\mathbf{1}_m]_h}{\partial \bar{\zeta}_j^*} = -\frac{\partial}{\partial \bar{\zeta}_j^*} \left(\sum_{\ell=1}^{m-1} e^{\frac{-C_{h\ell}+a_h+b_\ell}{\epsilon}} + e^{\frac{-C_{hm}+a_h}{\epsilon}} \right)$$
$$= -\frac{1}{\epsilon} \sum_{\ell=1}^{m-1} \delta_{\ell j} F_{h\ell} = -\frac{1}{\epsilon} F_{hj} = -\frac{1}{\epsilon}\Gamma_{hj}$$
$$\forall h = 1, \cdots, n, \quad j = 1, \cdots, m-1$$
$$\frac{\partial \psi(\xi^*, \bar{\zeta}^*, C)_\ell}{\partial \xi_i^*} = -\frac{\partial [\bar{F}^\top \mathbf{1}_n]_\ell}{\partial \xi_i^*} = -\frac{\partial}{\partial \xi_i^*} \sum_{h=1}^{n} e^{\frac{-C_{h\ell}+a_h+b_\ell}{\epsilon}}$$
$$= -\frac{1}{\epsilon} \sum_{h=1}^{n} \delta_{hi} F_{h\ell} = -\frac{1}{\epsilon} F_{i\ell} = -\frac{1}{\epsilon}\Gamma_{i\ell}$$
$$\forall \ell = 1, \cdots, m-1, \quad i = 1, \cdots, n$$
$$\frac{\partial \psi(\xi^*, \bar{\zeta}^*, C)_\ell}{\partial \bar{\zeta}_j^*} = -\frac{\partial [\bar{F}^\top \mathbf{1}_n]_\ell}{\partial \bar{\zeta}_j^*} = -\frac{\partial}{\partial \bar{\zeta}_j^*} \sum_{h=1}^{n} e^{\frac{-C_{h\ell}+a_h+b_\ell}{\epsilon}}$$
$$= -\frac{1}{\epsilon} \sum_{h=1}^{n} \delta_{\ell j} F_{h\ell} = -\frac{1}{\epsilon}\delta_{\ell j}\nu_\ell$$
$$\forall \ell = 1, \cdots, m-1, \quad j = 1, \cdots, m-1.$$

To sum up, we have

$$H = -\frac{1}{\epsilon}\begin{bmatrix} \mathrm{diag}(\mu) & \bar{\Gamma} \\ \bar{\Gamma}^T & \mathrm{diag}(\bar{\nu}) \end{bmatrix}.$$

Following the formula for inverse of block matrices,

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}(\mathbf{D}-\mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1}\mathbf{C}\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{B}(\mathbf{D}-\mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1} \\ -(\mathbf{D}-\mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1}\mathbf{C}\mathbf{A}^{-1} & (\mathbf{D}-\mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1} \end{bmatrix},$$

denote

$$\mathcal{K} = \mathrm{diag}(\bar{\nu}) - \bar{\Gamma}^T(\mathrm{diag}(\mu))^{-1}\bar{\Gamma}.$$

Note that $\mathcal{K}$ is just a scalar for SOFT top-k operator, and is a $(k-1)\times(k-1)$ matrix for sorted SOFT top-k operator. Therefore computing its inverse is not expensive. Finally we have

$$H^{-1} = -\epsilon\begin{bmatrix} (\mathrm{diag}(\mu))^{-1} + (\mathrm{diag}(\mu))^{-1}\bar{\Gamma}\mathcal{K}^{-1}\bar{\Gamma}^T(\mathrm{diag}(\mu))^{-1} & -(\mathrm{diag}(\mu))^{-1}\bar{\Gamma}\mathcal{K}^{-1} \\ -\mathcal{K}^{-1}\bar{\Gamma}^T(\mathrm{diag}(\mu))^{-1} & \mathcal{K}^{-1} \end{bmatrix}.$$

And also

$$D^{(1)}_{hij} = \frac{1}{\epsilon}\delta_{hi}\Gamma_{ij}$$
$$D^{(2)}_{\ell ij} = \frac{1}{\epsilon}\delta_{\ell j}\Gamma_{ij}.$$

The above derivation can actually be viewed as we explicitly force $b_m = 0$, i.e., no matter how C changes, b_m does not change. Therefore, we can treat $\frac{db_m}{dC} = \mathbf{0}_{n\times m}$, and we get the equation in the theorem. □

After we obtain $\frac{d\xi^*}{dC}$ and $\frac{d\zeta^*}{dC}$, we can now compute $\frac{d\Gamma}{dC}$.

$$\frac{d\Gamma_{h\ell}}{dC_{ij}} = \frac{d}{dC_{ij}} e^{\frac{-C_{h\ell}+a_h+b_\ell}{\epsilon}} = \frac{1}{\epsilon}\left(-\Gamma_{h\ell}\delta_{ih}\delta_{j\ell} + \Gamma_{h\ell}\frac{d\xi_h^*}{dC_{ij}} + \Gamma_{h\ell}\frac{db_\ell^*}{dC_{ij}}\right).$$

Finally, in the back-propagation step, we can compute the gradient of the loss L w.r.t. C,

$$\begin{aligned}
\frac{dL}{dC_{ij}} &= \sum_{h,\ell=1}^{n,m} \frac{dL}{d\Gamma_{h\ell}}\frac{d\Gamma_{h\ell}}{dC_{ij}} \\
&= \frac{1}{\epsilon}\left(-\sum_{h,\ell=1}^{n,m}\frac{dL}{d\Gamma_{h\ell}}\Gamma_{h\ell}\delta_{in}\delta_{j\ell} + \sum_{h,\ell=1}^{n,m}\frac{dL}{d\Gamma_{h\ell}}\Gamma_{h\ell}\frac{d\xi_h^*}{dC_{ij}} + \sum_{h,\ell=1}^{n,m}\frac{dL}{d\Gamma_{h\ell}}\Gamma_{h\ell}\frac{db_\ell^*}{dC_{ij}}\right) \\
&= \frac{1}{\epsilon}\left(-\frac{dL}{d\Gamma_{ij}}\Gamma_{ij} + \sum_{h,\ell=1}^{n,m}\frac{dL}{d\Gamma_{h\ell}}\Gamma_{h\ell}\frac{d\xi_h^*}{dC_{ij}} + \sum_{h,\ell=1}^{n,m}\frac{dL}{d\Gamma_{h\ell}}\Gamma_{h\ell}\frac{db_\ell^*}{dC_{ij}}\right).
\end{aligned}$$

We summarize the above procedure for computing the gradient for sorted SOFT top-k operator in Algorithm Algorithm 7. This naive implementation takes $\mathcal{O}(n^2k)$ complexity, which is not efficient. Therefore, we modify the algorithm using the associative law of matrix multiplications, so that the complexity is lowered to $\mathcal{O}(nk)$. We summarize the modified algorithm in Algorithm Algorithm 8.

We also include the `PyTorch` implementation of the forward pass and backward pass as shown below. The code is executed by creating an instance of `TopK_custom`, and the forward pass and the backward pass is run similar to any other `PyTorch` model.

Algorithm 7 Gradient for Sorted Top-K

Require: $C \in \mathbb{R}^{n\times(k+1)}, \mu \in \mathbb{R}^n, \nu \in \mathbb{R}^{k+1}, \frac{d\mathcal{L}}{d\Gamma} \in \mathbb{R}^{n\times(k+1)}, \epsilon$

Run forward pass to get Γ

$\bar{\nu} = \nu[:-1], \bar{\Gamma} = \Gamma[:,:-1]$

$\mathcal{K} \leftarrow \text{diag}(\bar{\nu}) - \bar{\Gamma}^T(\text{diag}(\mu))^{-1}\bar{\Gamma}$ # $\mathcal{K} \in \mathbb{R}^{k\times k}$

$H1 \leftarrow (\text{diag}(\mu))^{-1} + (\text{diag}(\mu))^{-1}\bar{\Gamma}\mathcal{K}^{-1}\bar{\Gamma}^T(\text{diag}(\mu))^{-1}$ # $H1 \in \mathbb{R}^{n\times n}$

$H2 \leftarrow -(\text{diag}(\mu))^{-1}\bar{\Gamma}\mathcal{K}^{-1}$ # $H2 \in \mathbb{R}^{n\times k}$

$H3 \leftarrow (H2)^T$ # $H3 \in \mathbb{R}^{k\times n}$

$H4 \leftarrow \mathcal{K}^{-1}$ # $H4 \in \mathbb{R}^{k\times k}$

Pad $H2$ to be $[n, k+1]$ in the last column with value 0

Pad $H4$ to be $[k, k+1]$ in the last column with value 0

$[\frac{d\xi^*}{dC}]_{hij} \leftarrow [H1]_{hi}\Gamma_{ij} + [H2]_{hj}\Gamma_{ij}$ # $\frac{d\xi^*}{dC} \in \mathbb{R}^{n\times n\times(k+1)}$

$[\frac{db^*}{dC}]_{\ell ij} \leftarrow [H3]_{\ell i}\Gamma_{ij} + [H4]_{\ell j}\Gamma_{ij}$ # $\frac{db^*}{dC} \in \mathbb{R}^{k\times n\times(k+1)}$

Pad $\frac{db^*}{dC}$ to be $[k+1, n, k+1]$ with value 0

$[\frac{d\mathcal{L}}{dC}]_{ij} \leftarrow \frac{1}{\epsilon}(-[\frac{d\mathcal{L}}{d\Gamma}]_{ij}\Gamma_{ij} + \sum_{h,\ell}[\frac{d\mathcal{L}}{d\Gamma}]_{h\ell}\Gamma_{h\ell}[\frac{d\xi^*}{dC}]_{hij} + \sum_{h,\ell}[\frac{d\mathcal{L}}{d\Gamma}]_{h\ell}\Gamma_{h\ell}[\frac{db^*}{dC}]_{\ell ij})$

Algorithm 8 Gradient for Sorted Top-k, with reduced memory

Require: $C \in \mathbb{R}^{N\times(K+1)}, \mu \in \mathbb{R}^N, \nu \in \mathbb{R}^{K+1}, \frac{d\mathcal{L}}{d\Gamma} \in \mathbb{R}^{N\times(K+1)}, \epsilon$

Run forward pass to get Γ

$\bar{\nu} = \nu[:-1], \bar{\Gamma} = \Gamma[:,:-1]$

$\mathcal{K} \leftarrow \text{diag}(\bar{\nu}) - \bar{\Gamma}^T(\text{diag}(\mu))^{-1}\bar{\Gamma}$ # $\mathcal{K} \in \mathbb{R}^{K\times K}$

$\mu'_i = \mu_i^{-1}$

$L \leftarrow (\text{diag}(\mu))^{-1}\bar{\Gamma}\mathcal{K}^{-1}$ # $L \in \mathbb{R}^{N\times K}$

$G1 \leftarrow \frac{d\mathcal{L}}{d\Gamma} \odot \Gamma$ # $G1 \in \mathbb{R}^{N\times K}$

$g1 \leftarrow [G1]\mathbf{1}_K$, $g2 \leftarrow [G1]^T\mathbf{1}_N$ # $g1 \in \mathbb{R}^N, g2 \in \mathbb{R}^K$

$G21 \leftarrow (g1 \odot \mu').expand\_dims(1) \odot \Gamma$ # $G21 \in \mathbb{R}^{N\times(K+1)}$

$G22 \leftarrow ((g1)^T L\bar{\Gamma}^T \odot \mu').expand\_dims(1) \odot \Gamma$ # $G22 \in \mathbb{R}^{N\times(K+1)}$

$G23 \leftarrow -((g1)^T L).pad\_last\_entry(0).expand\_dims(0) \odot \Gamma$ # $G23 \in \mathbb{R}^{N\times(K+1)}$

$G2 = G21 + G22 + G23$ # $G2 \in \mathbb{R}^{N\times(K+1)}$

$g2 \leftarrow g2[:-1]$

$G31 \leftarrow -(L(g2)).expand\_dims(1) \odot \Gamma$ # $G31 \in \mathbb{R}^{N\times(K+1)}$

$G32 \leftarrow (\mathcal{K}^{-1}(g2)).pad\_last\_entry(0).expand\_dims(0) \odot \Gamma$ # $G32 \in \mathbb{R}^{N\times(K+1)}$

$G3 = G31 + G32$ # $G3 \in \mathbb{R}^{N\times(K+1)}$

$\frac{d\mathcal{L}}{dC} \leftarrow \frac{1}{\epsilon}(-G1 + G2 + G3)$

```
def sinkhorn_forward(C, mu, nu, epsilon, max_iter):
    bs, n, k_ = C.size()

    v = torch.ones([bs, 1, k_])/(k_)
    G = torch.exp(-C/epsilon)
    if torch.cuda.is_available():
        v = v.cuda()

    for i in range(max_iter):
        u = mu/(G*v).sum(-1, keepdim=True)
        v = nu/(G*u).sum(-2, keepdim=True)

    Gamma = u*G*v
    return Gamma

def sinkhorn_forward_stablized(C, mu, nu, epsilon, max_iter):
    bs, n, k_ = C.size()
    k = k_-1

    f = torch.zeros([bs, n, 1])
    g = torch.zeros([bs, 1, k+1])
    if torch.cuda.is_available():
        f = f.cuda()
        g = g.cuda()

    epsilon_log_mu = epsilon*torch.log(mu)
    epsilon_log_nu = epsilon*torch.log(nu)

    def min_epsilon_row(Z, epsilon):
        return -epsilon*torch.logsumexp((-Z)/epsilon, -1, keepdim=True)

    def min_epsilon_col(Z, epsilon):
        return -epsilon*torch.logsumexp((-Z)/epsilon, -2, keepdim=True)

    for i in range(max_iter):
        f = min_epsilon_row(C-g, epsilon)+epsilon_log_mu
        g = min_epsilon_col(C-f, epsilon)+epsilon_log_nu

    Gamma = torch.exp((-C+f+g)/epsilon)
    return Gamma

def sinkhorn_backward(grad_output_Gamma, Gamma, mu, nu, epsilon):

    nu_ = nu[:,:,:-1]
    Gamma_ = Gamma[:,:,:-1]

    bs, n, k_ = Gamma.size()

    inv_mu = 1./(mu.view([1,-1]))  #[1, n]
    Kappa = torch.diag_embed(nu_.squeeze(-2)) \
            -torch.matmul(Gamma_.transpose(-1, -2) * inv_mu.unsqueeze(-2), Gamma_)    #[bs, k, k]

    inv_Kappa = torch.inverse(Kappa) #[bs, k, k]

    Gamma_mu = inv_mu.unsqueeze(-1)*Gamma_
    L = Gamma_mu.matmul(inv_Kappa) #[bs, n, k]
```

```
    G1 = grad_output_Gamma * Gamma #[bs, n, k+1]

    g1 = G1.sum(-1)
    G21 = (g1*inv_mu).unsqueeze(-1)*Gamma  #[bs, n, k+1]
    g1_L = g1.unsqueeze(-2).matmul(L)  #[bs, 1, k]
    G22 = g1_L.matmul(Gamma_mu.transpose(-1,-2)).transpose(-1,-2)*Gamma  #[bs, n, k+1]
    G23 = - F.pad(g1_L, pad=(0, 1), mode='constant', value=0)*Gamma  #[bs, n, k+1]
    G2 = G21 + G22 + G23  #[bs, n, k+1]

    del g1, G21, G22, G23, Gamma_mu

    g2 = G1.sum(-2).unsqueeze(-1) #[bs, k+1, 1]
    g2 = g2[:,:-1,:]  #[bs, k, 1]
    G31 = - L.matmul(g2)*Gamma  #[bs, n, k+1]
    G32 = F.pad(inv_Kappa.matmul(g2).transpose(-1,-2), pad=(0, 1), mode='constant', value=0)*Gamma  #[bs, n, k+1]
    G3 = G31 + G32  #[bs, n, k+1]

    grad_C = (-G1+G2+G3)/epsilon  #[bs, n, k+1]
    return grad_C

class TopKFunc(Function):
    @staticmethod
    def forward(ctx, C, mu, nu, epsilon, max_iter):

        with torch.no_grad():
            if epsilon>1e-2:
                Gamma = sinkhorn_forward(C, mu, nu, epsilon, max_iter)
                if bool(torch.any(Gamma!=Gamma)):
                    print('Nan appeared in Gamma, re-computing...')
                    Gamma = sinkhorn_forward_stablized(C, mu, nu, epsilon, max_iter)
            else:
                Gamma = sinkhorn_forward_stablized(C, mu, nu, epsilon, max_iter)
            ctx.save_for_backward(mu, nu, Gamma)
            ctx.epsilon = epsilon
        return Gamma

    @staticmethod
    def backward(ctx, grad_output_Gamma):

        epsilon = ctx.epsilon
        mu, nu, Gamma = ctx.saved_tensors
        # mu [1, n, 1]
        # nu [1, 1, k+1]
        #Gamma [bs, n, k+1]
        with torch.no_grad():
            grad_C = sinkhorn_backward(grad_output_Gamma, Gamma, mu, nu, epsilon)
        return grad_C, None, None, None, None


class TopK_custom(torch.nn.Module):
    def __init__(self, k, epsilon=0.1, max_iter = 200):
        super(TopK_custom1, self).__init__()
        self.k = k
        self.epsilon = epsilon
        self.anchors = torch.FloatTensor([k-i for i in range(k+1)]).view([1,1, k+1])
        self.max_iter = max_iter
```

```
        if torch.cuda.is_available():
            self.anchors = self.anchors.cuda()

    def forward(self, scores):
        bs, n = scores.size()
        scores = scores.view([bs, n, 1])

        #find the -inf value and replace it with the minimum value except -inf
        scores_ = scores.clone().detach()
        max_scores = torch.max(scores_).detach()
        scores_[scores_==float('-inf')] = float('inf')
        min_scores = torch.min(scores_).detach()
        filled_value = min_scores - (max_scores-min_scores)
        mask = scores==float('-inf')
        scores = scores.masked_fill(mask, filled_value)

        C = (scores-self.anchors)**2
        C = C / (C.max().detach())

        mu = torch.ones([1, n, 1], requires_grad=False)/n
        nu = [1./n for _ in range(self.k)]
        nu.append((n-self.k)/n)
        nu = torch.FloatTensor(nu).view([1, 1, self.k+1])

        if torch.cuda.is_available():
            mu = mu.cuda()
            nu = nu.cuda()

        Gamma = TopKFunc.apply(C, mu, nu, self.epsilon, self.max_iter)

        A = Gamma[:,:,:self.k]*n

        return A, None
```

C.3 Experiment Settings

C.3.1 kNN

The settings of the neural networks, the training procedure, and the number of neighbors k, and the tuning procedures are similar to [96]. The tuning o ϵ ranging from 10^{-6} to 10^{-2}. Other settings are shown in Table Table C.1.

Note that f_θ is a feature extraction neural network, so that model specified in the last row of Table Table C.1 does not contain the final activation layer and the linear layer.

Baselines. In the baselines, the results of kNN, kNN+PCA, kNN+AE, kNN+NeuralSort is

Table C.1: Parameter settings for kNN experiments.

Dataset	MNIST	CIFAR-10
k	9	9
ϵ	10^{-3}	10^{-5}
Batch size of query samples	100	100
Batch size of template samples	100	100
Optimizer	SGD	SGD
Learning rate	10^{-3}	10^{-3}
Momentum	0.9	0.9
Weight decay	5×10^{-4}	5×10^{-4}
Model	2-layer convolutional network	ResNet18

copied from [96]. The result of RelaxSubSample is copied from [97].

The implementation of kNN+[94] is based on [96]. Specifically, the outputs of the models in [94] and [96] are both doubly stochastic matrices. So in the implementation of kNN+[94], we adopt the algorithm in [96], except that we replace the module of computing the doubly stochastic matrix to be the one in [94]. We extensively tuned k, ϵ and the learning rate, but cannot achieve a better score for this experiment.

The baselines kNN+Softmax k times, kNN+pretrained CNN, and CE+CNN adopts the identical neural networks as our model. We remark that the scores reported in [96] for CNN+CE are 99.4% for MNIST and 95.1% for CIFAR-10. However, our experiments *using their code* cannot reproduce the reported scores: and the scores are 99.0% and 90.9%, respectively. Therefore, the reported score for MNIST is implemented by us, and the score for CIFAR-10 is copied from [78].

C.3.2 Beam Search

Algorithm. We now elaborate how to backtrack the predecessors $E^{(1:t),r}$ for an embedding $E^{(t+1),\ell}$, and how to compute the likelihood $\mathcal{L}_{\mathrm{s}}(E^{(1:t+1),\ell})$, which we have omitted in Algorithm Algorithm 5. Specifically, in standard beam search algorithm, each selected token $\widetilde{y}^{(t+1),\ell}$ is generated from a specific predecessor, and thus the backtracking is straightforward. In beam search with sorted SOFT top-k operator, however, each computed embedding $E^{(1:t),r}$ is a weighted sum of the output from all predecessors, so that it is not corre-

sponding to one specific predecessor. To address this difficulty, we select the predecessor for $E^{(t+1),\ell}$ with the largest weight, i.e.,

$$(o, r) = \underset{(j,i)}{\operatorname{argmax}} A^{(t),\epsilon}_{ji,\ell}.$$

This is a good approximation because $A^{(t),\epsilon}$ is a smoothed 0-1 tensor, i.e., for each ℓ, there is only one entry that is approximately 1 in $A^{(t),\epsilon}_{:,:,\ell}$, while the others are approximately 0. The likelihood is then computed as follows

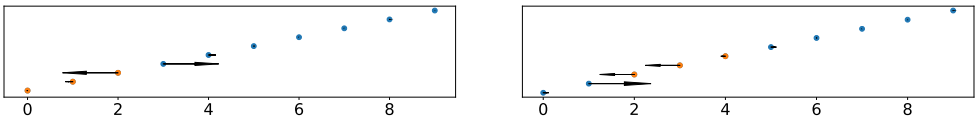
$$\mathcal{L}_{\mathrm{s}}(E^{(1:t+1),\ell}) = \mathcal{L}_{\mathrm{s}}(E^{(1:t),r})\mathbb{P}(y^{t+1} = \omega_o|\widetilde{h}^{(t),r}(E^{(1:t),r})).$$

Implementation. The implemented model is identical to [100]. Different from [100], here we also preprocess the data with *byte pair encoding* [101].

We adopt beam size 5, teacher forcing ratio $\rho = 0.8$, and $\epsilon = 10^{-1}$. The training procedure is as follows: We first pretrain the model with teacher forcing training procedure. The pretraining procedure has initial learning rate 1, learning rate decay 0.1 starting from iteration 5×10^5 for every 10^5 iterations. We pretrain it for 10^6 iterations in total. We then train the model using the combined training procedure for 10^5 iterations with learning rate 0.05.

C.3.3 Top-k Attention

The settings of the baseline model on data pre-processing, model, and the training procedure, evaluation procedure is identical to https://opennmt.net/OpenNMT-py/extended.html. The settings of the proposed model only differs in that we adopt SOFT top-k attention instead of the standard soft attention.



(a) $\mathcal{I} = \{0, 1, 2\}$. (b) $\mathcal{I} = \{2, 3, 4\}$.

Figure C.1: Illustration of the gradient of the SOFT top-k operators. The arrows represent the direction and magnitude of the gradient. The orange dots corresponds to the ground truth elements.

C.4 Visualization of the Gradients

In this section we visualize the computed gradient using a toy example mimicking the settings of kNN classification. Specifically, we input 10 scores computed from 10 images, i.e., $\mathcal{X} = \{0, 1, 2, \cdots, 9\}$, into the SOFT top-$k$ operator, and select the top-3 elements. Denote the indices of the images with the same labels as the query sample as $\mathcal{I}$. Similar to kNN classification, we want to maximize $\sum_{i \in \mathcal{I}} A_i^\epsilon$.

We visualize the gradient on $\mathcal{X}$ with respect to this objective function in Figure Figure C.1. In Figure Figure C.1a, $\mathcal{I}$ is the same as the indices of top-3 scores. In this case, the gradient will push the gap between the top-3 scores and the rest scores even further. In Figure Figure C.1b, $\mathcal{I}$ is different from the indices of top-3 scores. In this case, the scores corresponding to $\mathcal{I}$ are pushed to be smaller, while the others are pushed to be larger.

APPENDIX D

APPENDIX ON ROBOT

D.1 Connection between OT and RWOC

Theorem 1. Denote $\Pi(a,b) = \{S \in \mathbb{R}^{n\times m} : S\mathbf{1}_m = a, S^\top \mathbf{1}_n = b, S_{ij} \geq 0\}$ for any $a \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$. Then at least one of the optimal solutions of the following problem lies in $\mathcal{P}$.

$$\min_{S\in\mathbb{R}^{n\times n}} \langle C(w), S\rangle, \quad \text{s.t. } S \in \Pi(\mathbf{1}_n, \mathbf{1}_n). \tag{D.1}$$

Proof. Denote the optimal solution of (Equation D.1) as Z^*. As we mentioned earlier, this is a direct corollary of Birkhoff–von Neumann theorem [127, 128]. Specifically, Birkhoff–von Neumann theorem claims that the polytope $\Pi(\mathbf{1}_n, \mathbf{1}_n)$ is the convex hull of the set of $n \times n$ permutation matrices, and furthermore that the vertices of $\Pi(\mathbf{1}_n, \mathbf{1}_n)$ are precisely the permutation matrices.

On the other hand, (Equation D.1) is a linear optimization problem. There would be at least one optimal solutions lies at the vertices given the problem is feasible. As a result, there would be at least one Z^* being a permutation matrix. □

D.2 Proof of Proposition 4

The bilevel optimization formulation has a better gradient descent iteration complexity than alternating minimization. To see this, consider a quadratic function $F(a_1, a_2) = a^\top P a + b^\top a$, where $a_1 \in \mathbb{R}^{d_1}$, $a_2 \in \mathbb{R}^{d_2}$, $a = [a_1^\top, a_2^\top]^\top \in \mathbb{R}^{(d_1+d_2)}$, $P \in \mathbb{R}^{(d_1+d_2)\times(d_1+d_2)}$, $b \in \mathbb{R}^{(d_1+d_2)}$. To further simplify the discussion, we assume $P = \rho\mathbf{1}_{(d_1+d_2)}\mathbf{1}_{(d_1+d_2)}^\top + (1-\rho)I_{d_1+d_2}$, where $I_{d_1+d_2}$ is the identity matrix. Then we have the following proposition.

Proposition 1. Given F defined in (Equation 5.9), we have

$$\frac{\lambda_{\max}(\nabla^2 F(a_1))}{\lambda_{\min}(\nabla^2 F(a_1))} = 1 + \frac{1-\rho+\lambda}{d_2\rho-\rho+\lambda+1} \cdot \frac{d_1\rho}{1-\rho} \quad \text{and} \quad \frac{\lambda_{\max}(\nabla^2_{a_1a_1} L(a_1,a_2))}{\lambda_{\min}(\nabla^2_{a_1a_1} L(a_1,a_2))} = 1 + \frac{d_1\rho}{1-\rho}.$$

Proof. For alternating minimization, the Hessian for a_1 is a submatrix of P, i.e.,

$$H_{\mathrm{AM}} = \rho \mathbf{1}_{d_1}\mathbf{1}_{d_1}^\top + (1-\rho) I_{d_1},$$

whose condition number is

$$C_{\mathrm{AM}} = 1 + \frac{d_1\rho}{1-\rho}.$$

We now compute the condition number for ROBOT. Denote

$$P = \begin{bmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix},$$

where $P_{11} \in \mathbb{R}^{d_1 \times d_1}$, $P_{12} \in \mathbb{R}^{d_1 \times d_2}$, $P_{21} \in \mathbb{R}^{d_2 \times d_1}$, $P_{22} \in \mathbb{R}^{d_2 \times d_2}$, and $b_1 \in \mathbb{R}^{d_1}$, $b_2 \in \mathbb{R}^{d_2}$. ROBOT first minimize over a_2,

$$a_2^*(a_1) = \operatorname*{argmin}_{a_2} F(a_1,a_2) = -(P_{22}+\lambda I_{d_2})^{-1}(P_{21}a_1 + b_2/2).$$

Substituting $a_2^*(a_1)$ into $F(a_1,a_2)$, we can obtain the Hessian for a_1 is

$$H_{\mathrm{ROBOT}} = P_{11} - P_{12}(P_{22}+\lambda I_{d_2})^{-1}P_{21}.$$

Using Sherman–Morrison formula, we can explicitly express P_{22}^{-1} as

$$P_{22}^{-1} = \frac{1}{1-\rho+\lambda} I_{d_2} - \frac{\rho}{(1-\rho+\lambda)(1-\rho+\lambda+\rho d_2)} \mathbf{1}_{d_2}\mathbf{1}_{d_2}^\top.$$

Substituting it into H_{ROBOT},

$$H_{\mathrm{ROBOT}} = P_{11} - P_{12}P_{22}^{-1}P_{21} = (1-\rho)I_{d_1} + \left(\rho - \frac{d_2\rho^2}{d_2\rho - \rho + \lambda + 1}\right)\mathbf{1}_{d_1}\mathbf{1}_{d_1}^{\top}.$$

Therefore, the condition number is

$$C_{\mathrm{ROBOT}} = 1 + \frac{1-\rho+\lambda}{1-\rho}\frac{d_1\rho}{d_2\rho - \rho + \lambda + 1}.$$

□

Note that C_{AM} increases linearly with respect to d_1. Therefore, the optimization problem inevitably becomes ill-conditioned as dimension increase. In contrast, C_{ROBOT} can stay in the same order of magnitude when d_1 and d_2 increase simultaneously.

Since the iteration complexity of gradient descent is proportional to the condition number [165], ROBOT needs fewer iterations to converge than AM.

D.3 Differentiability

Theorem 2. For any $\epsilon > 0$, $S_\epsilon^*(w)$ is differentiable, as long as the cost $C(w)$ is differentiable with respect to w. As a result, the objective $\mathcal{L}_\epsilon(w) = \langle C(w), S_\epsilon^*(w)\rangle$ is also differentiable.

Proof. The proof is analogous to [166].

We first prove the differentiability of $S_\epsilon^*(w)$. This part of proof mirrors the proof in [89]. By Sinkhorn's scaling theorem [91],

$$S_\epsilon^*(w) = \mathrm{diag}(e^{\frac{\xi^*(w)}{\epsilon}})e^{-\frac{C(w)}{\epsilon}}\mathrm{diag}(e^{\frac{\zeta^*(w)}{\epsilon}}).$$

Therefore, since $C_{ij}(w)$ is differentiable, $\Gamma^{*,\epsilon}$ is differentiable if $(\xi^*(w), \zeta^*(w))$ is differentiable as a function of w.

Let us set

$$\mathcal{L}(\xi, \zeta; \mu, \nu, C) = \xi^T \mu + \zeta^T \nu - \epsilon \sum_{i,j=1}^{n,m} e^{-\frac{C_{ij} - \xi_i - \zeta_j}{\epsilon}}.$$

and recall that $(\xi^*, \zeta^*) = \text{argmax}_{\xi,\zeta} L(\xi, \zeta; \mu, \nu, C)$. The differentiability of (ξ^*, ζ^*) is proved using the Implicit Function theorem and follows from the differentiability and strict convexity in (ξ^*, ζ^*) of the function $\mathcal{L}$. □

Theorem 3. The gradient of $\mathcal{F}_\epsilon$ with respect to w is

$$\nabla_w \mathcal{F}_\epsilon = \frac{1}{\epsilon} \sum_{i,j=1}^{n,n} \left((1 - C_{ij}) S^*_{\epsilon,ij} + \sum_{h,\ell=1}^{n,n} C_{h\ell} S^*_{\epsilon,h\ell} \frac{d\xi^*_h}{dC_{ij}} + \sum_{h,\ell=1}^{n,n} C_{h\ell} S^*_{\epsilon,h\ell} \frac{d\zeta^*_\ell}{dC_{ij}} \right) \nabla_w C_{ij}, \tag{D.2}$$

$$\text{where} \begin{bmatrix} \nabla_C \xi^* \\ \nabla_C \zeta^* \end{bmatrix} = \begin{bmatrix} -H^{-1} D \\ \mathbf{0} \end{bmatrix} \quad \text{with} \quad -H^{-1} D \in \mathbb{R}^{(2n-1)\times n \times n}, \mathbf{0} \in \mathbb{R}^{1 \times n \times n},$$

$$D_{\ell ij} = \frac{1}{n\epsilon} \begin{cases} \delta_{\ell i} S^*_{\epsilon,ij}, & \ell = 1, \cdots, n; \\ \delta_{\ell j} S^*_{\epsilon,ij}, & \ell = n+1, \cdots, 2n-1, \end{cases} \qquad H^{-1} = -\epsilon n \begin{bmatrix} I_n + \bar{S}^*_\epsilon \mathcal{K}^{-1} \bar{S}^{*T}_\epsilon & -\bar{S}^*_\epsilon \mathcal{K}^{-1} \\ -\mathcal{K}^{-1} \bar{S}^{*T}_\epsilon & \mathcal{K}^{-1} \end{bmatrix},$$

$$\text{and} \quad \mathcal{K} = I_{n-1} - \bar{S}^{*T}_\epsilon \bar{S}^*_\epsilon, \quad \bar{S}^*_\epsilon = S^*_{\epsilon,1:n,1:n-1}.$$

Proof. This result is straightforward combining the Sinkhorn's scaling theorem and Theorem 3 in [166]. Specifically, notice the similarity between the lower-level optimization and (Equation 5.12),

$$\Gamma = \underset{\Gamma \in \Pi(\mu,\nu)}{\text{argmin}} \langle C(w), \Gamma \rangle + \epsilon \sum_{i,j} \Gamma_{ij} \ln \Gamma_{ij}.$$

We will first derive $\nabla_C \Gamma$, then derive $\nabla_w \mathcal{F}_\epsilon$ using the chain rule. To avoid possible confusion, we will derive the case where $\mu \in \mathbb{R}^n$ and $\nu \in \mathbb{R}^m$, and then take $\mu = \nu = \mathbf{1}_n/n$. The dual problem of the above optimization problem is

$$\xi^*, \zeta^* = \underset{\xi,\zeta}{\operatorname{argmax}} \mathcal{L}(\xi, \zeta; C),$$

where

$$\mathcal{L}(\xi, \zeta; C) = \xi^\top \mu + \zeta^\top \nu - \epsilon \sum_{i,j=1}^{n,m} e^{-\frac{C_{ij} - \xi_i - \zeta_j}{\epsilon}}.$$

And it is connected to the prime form by

$$\Gamma^{*,\epsilon} = \operatorname{diag}(e^{\frac{\xi^*}{\epsilon}}) e^{-\frac{C}{\epsilon}} \operatorname{diag}(e^{\frac{\zeta^*}{\epsilon}}).$$

Notice that there is one redundant dual variable, since $\mu \mathbf{1}_n = \nu \mathbf{1}_m = 1$. Therefore, we can rewrite $\mathcal{L}(\xi, \zeta; C)$ as

$$\mathcal{L}(\xi, \bar{\zeta}; C) = \xi^T \mu + \bar{\zeta}^T \bar{\nu} - \epsilon \sum_{i,j=1}^{n,m-1} e^{\frac{-C_{ij} + \xi_i + \zeta_j}{\epsilon}} - \epsilon \sum_{i=1}^{n} e^{\frac{-C_{im} + \xi_i}{\epsilon}}.$$

Denote

$$\phi(\xi, \bar{\zeta}, C) = \frac{d\mathcal{L}(\xi, \bar{\zeta}; C)}{d\xi} = \mu - F\mathbf{1}_m, \tag{D.3}$$

$$\psi(\xi, \bar{\zeta}, C) = \frac{d\mathcal{L}(\xi, \bar{\zeta}; C)}{d\bar{\zeta}} = \bar{\nu} - \bar{F}^\top \mathbf{1}_n, \tag{D.4}$$

where

$$
\begin{aligned}
F_{ij} &= e^{\frac{-C_{ij}+\xi_i+\zeta_j}{\epsilon}}, \quad \forall i = 1, \cdots, n, \quad j = 1, \cdots, m-1 \\
F_{im} &= e^{\frac{-C_{im}+\xi_i}{\epsilon}}, \quad \forall i = 1, \cdots, n, \\
\bar{F} &= F_{:,:-1}.
\end{aligned}
$$

Since $(\xi^*, \bar{\zeta}^*)$ is a maximum of $\mathcal{L}(\xi, \bar{\zeta}; C)$, we have

$$
\begin{aligned}
\phi(\xi^*, \bar{\zeta}^*, C) &= 0, \\
\psi(\xi^*, \bar{\zeta}^*, C) &= 0.
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
\frac{d\phi(\xi^*, \bar{\zeta}^*, C)}{dC} &= \frac{\partial \phi(\xi^*, \bar{\zeta}^*, C)}{\partial C} + \frac{\partial \phi(\xi^*, \bar{\zeta}^*, C)}{\partial \xi^*}\frac{d\xi^*}{dC} + \frac{\partial \phi(\xi^*, \bar{\zeta}^*, C)}{\partial \bar{\zeta}^*}\frac{d\bar{\zeta}^*}{dC} = 0, \\
\frac{d\psi(\xi^*, \bar{\zeta}^*, C)}{dC} &= \frac{\partial \psi(\xi^*, \bar{\zeta}^*, C)}{\partial C} + \frac{\partial \psi(\xi^*, \bar{\zeta}^*, C)}{\partial \xi^*}\frac{d\xi^*}{dC} + \frac{\partial \psi(\xi^*, \bar{\zeta}^*, C)}{\partial \bar{\zeta}^*}\frac{d\bar{\zeta}^*}{dC} = 0.
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
\begin{bmatrix} \frac{d\xi^*}{dC} \\ \frac{d\bar{\zeta}^*}{dC} \end{bmatrix} &= - \begin{bmatrix} \frac{\partial \phi(\xi^*, \bar{\zeta}^*, C)}{\partial \xi^*} & \frac{\partial \phi(\xi^*, \bar{\zeta}^*, C)}{\partial \bar{\zeta}^*} \\ \frac{\partial \psi(\xi^*, \bar{\zeta}^*, C)}{\partial \xi^*} & \frac{\partial \psi(\xi^*, \bar{\zeta}^*, C)}{\partial \bar{\zeta}^*} \end{bmatrix}^{-1} \begin{bmatrix} \frac{\partial \phi(\xi^*, \bar{\zeta}^*, C)}{\partial C} \\ \frac{\partial \psi(\xi^*, \bar{\zeta}^*, C)}{\partial C} \end{bmatrix} \\
&\triangleq -H^{-1} \begin{bmatrix} D^{(1)} \\ D^{(2)} \end{bmatrix} \\
&\triangleq -H^{-1} D.
\end{aligned}
$$

After some derivations, we have

$$H = -\frac{1}{\epsilon}\begin{bmatrix} \mathrm{diag}(\mu) & \bar{\Gamma} \\ \bar{\Gamma}^T & \mathrm{diag}(\bar{\nu}) \end{bmatrix}.$$

Following the formula for inverse of block matrices,

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1}\mathbf{C}\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{B}(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1} \\ -(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1}\mathbf{C}\mathbf{A}^{-1} & (\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1} \end{bmatrix},$$

denote

$$\mathcal{K} = \mathrm{diag}(\bar{\nu}) - \bar{\Gamma}^T(\mathrm{diag}(\mu))^{-1}\bar{\Gamma}.$$

Finally we have

$$H^{-1} = -\epsilon\begin{bmatrix} (\mathrm{diag}(\mu))^{-1} + (\mathrm{diag}(\mu))^{-1}\bar{\Gamma}\mathcal{K}^{-1}\bar{\Gamma}^T(\mathrm{diag}(\mu))^{-1} & -(\mathrm{diag}(\mu))^{-1}\bar{\Gamma}\mathcal{K}^{-1} \\ -\mathcal{K}^{-1}\bar{\Gamma}^T(\mathrm{diag}(\mu))^{-1} & \mathcal{K}^{-1} \end{bmatrix}.$$

And also

$$D^{(1)}_{hij} = \frac{1}{\epsilon}\delta_{hi}\Gamma_{ij}$$
$$D^{(2)}_{\ell ij} = \frac{1}{\epsilon}\delta_{\ell j}\Gamma_{ij}.$$

The above derivation can actually be viewed as we explicitly force $\zeta_m = 0$, i.e., no matter how C changes, ζ_m does not change. Therefore, we can treat $\frac{d\zeta_m}{dC} = \mathbf{0}_{n\times m}$.

After we obtain $\frac{d\xi^*}{dC}$ and $\frac{d\zeta^*}{dC}$, we can now compute $\frac{d\Gamma}{dC}$.

$$\frac{d\Gamma_{h\ell}}{dC_{ij}} = \frac{d}{dC_{ij}}e^{\frac{-C_{h\ell}+\xi^*_h+\zeta^*_\ell}{\epsilon}} = \frac{1}{\epsilon}\left(-\Gamma_{h\ell}\delta_{ih}\delta_{j\ell} + \Gamma_{h\ell}\frac{d\xi^*_h}{dC_{ij}} + \Gamma_{h\ell}\frac{d\zeta^*_\ell}{dC_{ij}}\right).$$

Note that $\mathcal{F}_\epsilon(w) = \langle C(w), n\Gamma \rangle$. Substituting $\frac{d\Gamma_{h\ell}}{dC_{ij}}$ into the expression of $\nabla_w \mathcal{F}_\epsilon(w)$, we get the equation in the theorem.

□

D.4 Algorithm of the Forward Pass for ROBOT-robust

For better numerical stability, in practice we add two more regularization terms,

$$\begin{aligned} S_{\mathrm{r}}^*(w), \bar{\mu}^*, \bar{\nu}^* = \ & \mathrm{argmin}_{S \in \Pi(\bar{\mu}, \bar{\nu}),\ \bar{\mu}, \bar{\nu} \in \Delta_n} \langle C(w), S \rangle + \epsilon H(S) + \epsilon_1 h(\bar{\mu}) + \epsilon_2 h(\bar{\nu}), \quad \text{(D.5)} \\ & \text{s.t. } \mathcal{F}(\bar{\mu}, \mu) \leq \rho_1,\ \mathcal{F}(\bar{\nu}, \nu) \leq \rho_2, \end{aligned}$$

where $h(\bar{\mu}) = \sum_i \bar{\mu}_i \log \bar{\mu}_i$ is the entropy function for vectors. This can avoid the entries of $\bar{\mu}$ and $\bar{\nu}$ shrink to zeros when updated by gradient descent. We remark that since we have entropy term $H(S)$, the entries of S would not be exactly zeros. Furthermore, we have $\bar{\mu} = S\mathbf{1}$ and $\bar{\mu} = S\mathbf{1}$. Therefore, theoretically the entries of $\bar{\mu}$ and $\bar{\nu}$ will not be zeros. We only add the two more entropy terms for numerical consideration. The detailed algorithm is in Algorithm Algorithm 9. Although the algorithm is not guaranteed to converge to a feasible solution, in practice it usually converges to a good solution [167].

D.5 Algorithm of the Backward Pass for ROBOT-robust

We first summarize the outline of the derivation, then provide the detailed derivation.

D.5.1 Summary

Given $\bar{\mu}^*, \bar{\nu}^*, S_{\mathrm{r}}^*(w)$, we compute the Jacobian matrix $dS_{\mathrm{r}}^*(w)/dw$ using implicit differentiation and differentiable programming techinques. Specifically, the Lagrangian function

Algorithm 9 Solving S_{r}^* for robust matching

Require: $C \in \mathbb{R}^{m \times n}, \mu, \nu, K, \epsilon, L, \eta$
  $G_{ij} = e^{-\frac{C_{ij}}{\epsilon}}$
  $\bar{\mu} = \mu, \bar{\nu} = \nu$
  $b = \mathbf{1}_n$
  for $l = 1, \cdots, L$ **do**
    $a = \bar{\mu}/(Gb), b = \bar{\nu}/(G^T a)$
    $\bar{\mu} = \bar{\mu} - \eta(e^{\frac{a}{\epsilon}} + \epsilon_1 * \log \bar{\mu}), \bar{\nu} = \bar{\nu} - \eta(e^{\frac{b}{\epsilon}} + \epsilon_2 * \log \bar{\nu})$
    $\bar{\mu} = \max\{\bar{\mu}, 0\}, \bar{\nu} = \max\{\bar{\nu}, 0\}$
    $\bar{\mu} = \bar{\mu}/(\bar{\mu}^\top \mathbf{1}), \bar{\nu} = \bar{\nu}/(\bar{\nu}^\top \mathbf{1})$
    if $\|\bar{\mu} - \mu\|_2^2 > \rho_1$ **then**
      $\bar{\mu} = \mu + \sqrt{\rho_1} \frac{\bar{\mu} - \mu}{\|\bar{\mu} - \mu\|_2}$
    end if
    if $\|\bar{\nu} - \nu\|_2^2 > \rho_2$ **then**
      $\bar{\nu} = \nu + \sqrt{\rho_2} \frac{\bar{\nu} - \nu}{\|\bar{\nu} - \nu\|_2}$
    end if
  end for
  $S = \mathrm{diag}(a) \odot G \odot \mathrm{diag}(b)$

of Problem (Equation D.5) is

$$\begin{aligned}\mathcal{L} =& \langle C, S \rangle + \epsilon H(S) + \epsilon_1 h(\bar{\mu}) + \epsilon_2 h(\bar{\nu}) - \xi^\top (\Gamma \mathbf{1}_m - \mu) - \zeta^\top (\Gamma^\top \mathbf{1}_n - \nu) \\ &+ \lambda_1 (\bar{\mu}^\top \mathbf{1}_n - 1) + \lambda_2 (\bar{\nu}^\top \mathbf{1}_m - 1) + \lambda_3 (\|\bar{\mu} - \mu\|_2^2 - \rho_1) + \lambda_4 (\|\bar{\nu} - \nu\|_2^2 - \rho_2).\end{aligned}$$

where ξ and ζ are dual variables. The KKT conditions (Stationarity condition) imply that the optimal solution $\Gamma^{*,\epsilon}$ can be formulated using the optimal dual variables ξ^* and ζ^* as,

$$S_{\mathrm{r}}^* = \mathrm{diag}(e^{\frac{\xi^*}{\epsilon}}) e^{-\frac{C}{\epsilon}} \mathrm{diag}(e^{\frac{\zeta^*}{\epsilon}}). \tag{D.6}$$

By the chain rule, we have

$$\frac{dS_{\mathrm{r}}^*}{dw} = \frac{dS_{\mathrm{r}}^*}{dC} \frac{dC}{dw} = \left(\frac{\partial S_{\mathrm{r}}^*}{\partial C} + \frac{\partial S_{\mathrm{r}}^*}{\partial \xi^*} \frac{d\xi^*}{dC} + \frac{\partial S_{\mathrm{r}}^*}{\partial \zeta^*} \frac{d\zeta^*}{dC} \right) \frac{dC}{dw}.$$

Therefore, we can compute $dS_{\mathrm{r}}^*(w)/dw$ if we obtain $\frac{d\xi^*}{dC}$ and $\frac{d\zeta^*}{dC}$.

Substituting (Equation D.6) into the Lagrangian function, at the optimal solutions we

obtain

$$\mathcal{L} = \mathcal{L}(\xi^*, \zeta^*, \bar{\mu}^*, \bar{\nu}^*, \lambda_1^*, \lambda_2^*, \lambda_3^*, \lambda_4^*; C).$$

Denote $r^* = [(\xi^*)^\top, (\zeta^*)^\top, (\bar{\mu})^\top, (\bar{\nu})^\top, \lambda_1^*, \lambda_2^*, \lambda_3^*, \lambda_4^*]^\top$, and $\phi(r^*; C) = \partial\mathcal{L}(r^*; C)/\partial r^*$. At the optimal dual variable r^*, the KKT condition immediately yields $\phi(r^*; C) \equiv 0$. By the chain rule, we have

$$\frac{d\phi(r^*; C)}{dC} = \frac{\partial\phi(r^*; C)}{\partial C} + \frac{\partial\phi(r^*; C)}{\partial r^*}\frac{dr^*}{dC} = 0. \tag{D.7}$$

Rerranging terms, we obtain

$$\frac{dr^*}{dC} = -\left(\frac{\partial\phi(r^*; C)}{\partial r^*}\right)^{-1}\frac{\partial\phi(r^*; C)}{\partial C}. \tag{D.8}$$

Combining (Equation D.6), (Equation D.7), and (Equation D.8), we can then obtain $dS_{\mathrm{r}}^*(w)/dw$.

D.5.2 Details

Now we provide the detailed derivation for computing dS_{r}^*/dw.

Since S_{r}^* is the optimal solution of an optimization problem, we can follow the implicit function theorem to solve for the closed-form expression of the gradient. Specifically, we adopt $\mathcal{F}(\bar{\mu}, \nu) = \sum_i(\bar{\mu}_i - \mu_i)^2$, and rewrite the optimization problem as

$$\begin{aligned}
&\min_{\bar{\mu},\bar{\nu},S} \langle C, S\rangle + \epsilon\sum_{ij} S_{ij}(\log S_{ij} - 1) + \epsilon_1\sum_i \bar{\mu}_i(\log\bar{\mu}_i - 1) + \epsilon_2\sum_j \bar{\nu}_j(\log\bar{\nu}_j - 1),\\
\text{s.t.,}\ &\sum_j S_{ij} = \bar{\mu}_i, \quad \sum_i S_{ij} = \bar{\nu}_j,\\
&\sum_i \bar{\mu}_i = 1, \quad \sum_j \bar{\nu}_j = 1,\\
&\sum_i(\bar{\mu}_i - \mu_i)^2 \le \rho_1, \quad \sum_j(\bar{\nu}_j - \nu_j)^2 \le \rho_2.
\end{aligned}$$

The Language of the above problem is

$$
\begin{aligned}
&\mathcal{L}(C, S, \bar{\mu}, \bar{\nu}, \xi, \zeta, \lambda_1, \lambda_2, \lambda_3, \lambda_4) \\
&= \langle C, S \rangle + \epsilon \sum_{ij} S_{ij}(\log S_{ij} - 1) + \epsilon_1 \sum_i \bar{\mu}_i(\log \bar{\mu}_i - 1) + \epsilon_2 \sum_j \bar{\nu}_j(\log \bar{\nu}_j - 1) \\
&- \xi^\top (S\mathbf{1}_m - \bar{\mu}) - \zeta^\top (S^\top \mathbf{1}_n - \bar{\nu}) \\
&+ \lambda_1(\sum_i \bar{\mu}_i - 1) + \lambda_2(\sum_j \bar{\nu}_j - 1) + \lambda_3(\sum_i (\bar{\mu}_i - \mu_i)^2 - \rho_1) + \lambda_4(\sum_j (\bar{\nu}_j - \nu_j)^2 - \rho_2).
\end{aligned}
$$

Easy to see that the Slater's condition holds. Denote

$$
\mathcal{L}^* = \mathcal{L}(C, S_{\mathrm{r}}^*, \bar{\mu}^*, \bar{\nu}^*, \xi^*, \zeta^*, \lambda_1^*, \lambda_2^*, \lambda_3^*, \lambda_4^*).
$$

Following the KKT conditions,

$$
\frac{d\mathcal{L}^*}{dS_{\mathrm{r},ij}^*} = C_{ij} + \epsilon \log S_{\mathrm{r},ij}^* - \xi_i^* - \zeta_j^* = 0.
$$

Therefore, $S_{\mathrm{r},ij}^* = e^{\frac{\xi_i^* + \zeta_j^* - C_{ij}}{\epsilon}}$. Then we have

$$
\frac{dS_{\mathrm{r}}^*}{dw} = (\frac{\partial S_{\mathrm{r}}^*}{\partial C} + \frac{\partial S_{\mathrm{r}}^*}{\partial \xi^*}\frac{d\xi^*}{dC} + \frac{\partial S_{\mathrm{r}}^*}{\partial \zeta^*}\frac{d\zeta^*}{dC})\frac{dC}{dw}.
$$

So all we need to do is to compute $\frac{d\xi^*}{dC}$ and $\frac{d\zeta^*}{dC}$. Denote $F_{ij} = e^{\frac{\xi_i+\zeta_j-C_{ij}}{\epsilon}}$. Denote

$$
\begin{aligned}
\phi &= \frac{d\mathcal{L}}{d\xi} = \bar{\mu} - F\mathbf{1}_m, \\
\psi &= \frac{d\mathcal{L}}{d\zeta} = \bar{\nu} - F^\top \mathbf{1}_n, \\
p &= \frac{d\mathcal{L}}{d\bar{\mu}} = \xi + \lambda_1 \mathbf{1}_n + 2\lambda_3(\bar{\mu} - \mu) + \epsilon_1 \log \bar{\mu}, \\
q &= \frac{d\mathcal{L}}{d\bar{\nu}} = \zeta + \lambda_2 \mathbf{1}_m + 2\lambda_4(\bar{\nu} - \nu) + \epsilon_2 \log \bar{\nu}, \\
\chi_1 &= \frac{d\mathcal{L}}{d\lambda_1} = \bar{\mu}^\top \mathbf{1}_n - 1, \\
\chi_2 &= \frac{d\mathcal{L}}{d\lambda_2} = \bar{\nu}^\top \mathbf{1}_m - 1, \\
\chi_3 &= \lambda_3(\|\bar{\mu} - \mu\|_2^2 - \rho_1), \\
\chi_4 &= \lambda_4(\|\bar{\nu} - \nu\|_2^2 - \rho_2).
\end{aligned}
$$

Denote $\chi = [\chi_1, \chi_2, \chi_3, \chi_4]$, and $\lambda = [\lambda_1, \lambda_2, \lambda_3, \lambda_4]$. Following the KKT conditions, we have

$$
\phi = 0, \psi = 0, p = 0, q = 0, \chi = 0,
$$

at the optimal solutions. Therefore, for the optimal solutions we have

$$
\begin{aligned}
\frac{d\phi}{dC} &= \frac{\partial \phi}{\partial C} + \frac{\partial \phi}{\partial \xi^*}\frac{d\xi^*}{dC} + \frac{\partial \phi}{\partial \zeta^*}\frac{d\zeta^*}{dC} + \frac{\partial \phi}{\partial \bar{\mu}^*}\frac{d\bar{\mu}^*}{dC} + \frac{\partial \phi}{\partial \bar{\nu}^*}\frac{d\bar{\nu}^*}{dC} + \frac{\partial \phi}{\partial \lambda^*}\frac{d\lambda^*}{dC} = 0, \\
\frac{d\psi}{dC} &= \frac{\partial \psi}{\partial C} + \frac{\partial \psi}{\partial \xi^*}\frac{d\xi^*}{dC} + \frac{\partial \psi}{\partial \zeta^*}\frac{d\zeta^*}{dC} + \frac{\partial \psi}{\partial \bar{\mu}^*}\frac{d\bar{\mu}^*}{dC} + \frac{\partial \psi}{\partial \bar{\nu}^*}\frac{d\bar{\nu}^*}{dC} + \frac{\partial \psi}{\partial \lambda^*}\frac{d\lambda^*}{dC} = 0, \\
\frac{dp}{dC} &= \frac{\partial p}{\partial C} + \frac{\partial p}{\partial \xi^*}\frac{d\xi^*}{dC} + \frac{\partial p}{\partial \zeta^*}\frac{d\zeta^*}{dC} + \frac{\partial p}{\partial \bar{\mu}^*}\frac{d\bar{\mu}^*}{dC} + \frac{\partial p}{\partial \bar{\nu}^*}\frac{d\bar{\nu}^*}{dC} + \frac{\partial p}{\partial \lambda^*}\frac{d\lambda^*}{dC} = 0, \\
\frac{dq}{dC} &= \frac{\partial q}{\partial C} + \frac{\partial q}{\partial \xi^*}\frac{d\xi^*}{dC} + \frac{\partial q}{\partial \zeta^*}\frac{d\zeta^*}{dC} + \frac{\partial q}{\partial \bar{\mu}^*}\frac{d\bar{\mu}^*}{dC} + \frac{\partial q}{\partial \bar{\nu}^*}\frac{d\bar{\nu}^*}{dC} + \frac{\partial q}{\partial \lambda^*}\frac{d\lambda^*}{dC} = 0 \\
\frac{d\chi}{dC} &= \frac{\partial \chi}{\partial C} + \frac{\partial \chi}{\partial \xi^*}\frac{d\xi^*}{dC} + \frac{\partial \chi}{\partial \zeta^*}\frac{d\zeta^*}{dC} + \frac{\partial \chi}{\partial \bar{\mu}^*}\frac{d\bar{\mu}^*}{dC} + \frac{\partial \chi}{\partial \bar{\nu}^*}\frac{d\bar{\nu}^*}{dC} + \frac{\partial \chi}{\partial \lambda^*}\frac{d\lambda^*}{dC} = 0.
\end{aligned}
$$

Therefore, we have

$$\begin{bmatrix} \frac{d\xi^*}{dC} \\ \frac{d\zeta^*}{dC} \\ \frac{d\bar{\mu}^*}{dC} \\ \frac{d\bar{\nu}^*}{dC} \\ \frac{d\lambda^*}{dC} \end{bmatrix} = - \begin{bmatrix} \frac{\partial\phi}{\partial\xi^*} & \frac{\partial\phi}{\partial\zeta^*} & \frac{\partial\phi}{\partial\bar{\mu}^*} & \frac{\partial\phi}{\partial\bar{\nu}^*} & \frac{\partial\phi}{\partial\lambda^*} \\ \frac{\partial\psi}{\partial\xi^*} & \frac{\partial\psi}{\partial\zeta^*} & \frac{\partial\psi}{\partial\bar{\mu}^*} & \frac{\partial\psi}{\partial\bar{\nu}^*} & \frac{\partial\psi}{\partial\lambda^*} \\ \frac{\partial p}{\partial\xi^*} & \frac{\partial p}{\partial\zeta^*} & \frac{\partial p}{\partial\bar{\mu}^*} & \frac{\partial p}{\partial\bar{\nu}^*} & \frac{\partial p}{\partial\lambda^*} \\ \frac{\partial q}{\partial\xi^*} & \frac{\partial q}{\partial\zeta^*} & \frac{\partial q}{\partial\bar{\mu}^*} & \frac{\partial q}{\partial\bar{\nu}^*} & \frac{\partial q}{\partial\lambda^*} \\ \frac{\partial\chi}{\partial\xi^*} & \frac{\partial\chi}{\partial\zeta^*} & \frac{\partial\chi}{\partial\bar{\mu}^*} & \frac{\partial\chi}{\partial\bar{\nu}^*} & \frac{\partial\chi}{\partial\lambda^*} \end{bmatrix}^{-1} \begin{bmatrix} \frac{\partial\phi}{\partial C} \\ \frac{\partial\psi}{\partial C} \\ \frac{\partial p}{\partial C} \\ \frac{\partial q}{\partial C} \\ \frac{\partial\chi}{\partial C} \end{bmatrix}.$$

After some derivation, we have

$$\begin{bmatrix} \frac{d\xi^*}{dC} \\ \frac{d\zeta^*}{dC} \\ \frac{d\bar{\mu}^*}{dC} \\ \frac{d\bar{\nu}^*}{dC} \\ \frac{d\lambda_1^*}{dC} \\ \frac{d\lambda_2^*}{dC} \\ \frac{d\lambda_3^*}{dC} \\ \frac{d\lambda_4^*}{dC} \end{bmatrix} = - \begin{bmatrix} -\frac{1}{\epsilon}\mathrm{diag}(\bar{\mu}) & -\frac{1}{\epsilon}S_{\mathrm{r}}^* & \boldsymbol{I}_n & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ -\frac{1}{\epsilon}(S_{\mathrm{r}}^*)^\top & -\frac{1}{\epsilon}\mathrm{diag}(\bar{\nu}) & \mathbf{0} & \boldsymbol{I}_m & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{I}_n & \mathbf{0} & 2\lambda_3\boldsymbol{I}_n + \mathrm{diag}(\frac{\epsilon_1}{\bar{\mu}}) & \mathbf{0} & \mathbf{1}_n & \mathbf{0} & 2(\bar{\mu}-\mu) & \mathbf{0} \\ \mathbf{0} & \boldsymbol{I}_m & \mathbf{0} & 2\lambda_4\boldsymbol{I}_m + \mathrm{diag}(\frac{\epsilon_2}{\bar{\nu}}) & \mathbf{0} & \mathbf{1}_m & \mathbf{0} & 2(\bar{\nu}-\nu) \\ \mathbf{0} & \mathbf{0} & \mathbf{1}_n^\top & \mathbf{0} & 0 & 0 & 0 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1}_m^\top & 0 & 0 & 0 & 0 \\ \mathbf{0} & \mathbf{0} & 2\lambda_3(\bar{\mu}-\mu)^\top & \mathbf{0} & 0 & 0 & \|\bar{\mu}-\mu\|_2^2-\rho_1 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & 2\lambda_4(\bar{\nu}-\nu)^\top & 0 & 0 & 0 & \|\bar{\nu}-\nu\|_2^2-\rho_2 \end{bmatrix}^{-1} \begin{bmatrix} \frac{\partial\phi}{\partial C} \\ \frac{\partial\psi}{\partial C} \\ \mathbf{0} \\ \mathbf{0} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

and

$$\frac{\partial\phi_h}{\partial C_{ij}} = \frac{1}{\epsilon}\delta_{hi}S_{ij}, \forall h = 1, \cdots, n, \quad i = 1, \cdots, n, \quad j = 1, \cdots, m$$
$$\frac{\partial\psi_\ell}{\partial C_{ij}} = \frac{1}{\epsilon}\delta_{\ell j}S_{ij}, \forall \ell = 1, \cdots, m-1, \quad i = 1, \cdots, n, \quad j = 1, \cdots, m.$$

To efficiently solve for the inverse in the above equations, we denote

$$A = \begin{bmatrix} -\frac{1}{\epsilon}\mathrm{diag}(\bar{\mu}) & -\frac{1}{\epsilon}S_{\mathrm{r}}^* & \boldsymbol{I}_n & \mathbf{0} \\ -\frac{1}{\epsilon}(S_{\mathrm{r}}^*)^\top & -\frac{1}{\epsilon}\mathrm{diag}(\bar{\nu}) & \mathbf{0} & \boldsymbol{I}_m \\ \boldsymbol{I}_n & \mathbf{0} & 2\lambda_3\boldsymbol{I}_n + \mathrm{diag}(\frac{\epsilon_1}{\bar{\mu}}) & \mathbf{0} \\ \mathbf{0} & \boldsymbol{I}_m & \mathbf{0} & 2\lambda_4\boldsymbol{I}_m + \mathrm{diag}(\frac{\epsilon_2}{\bar{\nu}}) \end{bmatrix},$$

$$B_1 = \begin{bmatrix} \mathbf{1}_n & \mathbf{0} & 2(\bar{\mu}-\mu) & \mathbf{0} \\ \mathbf{0} & \mathbf{1}_m & \mathbf{0} & 2(\bar{\nu}-\nu) \end{bmatrix},$$

$$C_1 = \begin{bmatrix} \mathbf{1}_n^\top & \mathbf{0} \\ \mathbf{0} & \mathbf{1}_m^\top \\ 2\lambda_3(\bar{\mu}-\mu)^\top & \mathbf{0} \\ \mathbf{0} & 2\lambda_4(\bar{\nu}-\nu)^\top \end{bmatrix},$$

$$D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \|\bar{\mu}-\mu\|_2^2 - \rho_1 & 0 \\ 0 & 0 & 0 & \|\bar{\nu}-\nu\|_2^2 - \rho_2 \end{bmatrix}.$$

We first A^{-1} using the rules for inverting a block matrix,

$$A^{-1} = \begin{bmatrix} K & -KL \\ -LK & L+LKL \end{bmatrix} =: \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix}$$

where

$$L = \begin{bmatrix} 2\lambda_3 \boldsymbol{I}_n + \mathrm{diag}(\frac{\epsilon_1}{\bar{\mu}}) & \mathbf{0} \\ \mathbf{0} & 2\lambda_4 \boldsymbol{I}_m + \mathrm{diag}(\frac{\epsilon_1}{\bar{\nu}}) \end{bmatrix}^{-1}, \quad K = \left(\frac{1}{\epsilon} \begin{bmatrix} \mathrm{diag}(\bar{\mu}) & S_{\mathrm{r}}^* \\ (S_{\mathrm{r}}^*)^\top & \mathrm{diag}(\bar{\nu}) \end{bmatrix} + L \right)^{-1}.$$

Then using the rules of inverting a block matrix again, we have

$$\begin{bmatrix} \dfrac{d\xi^*}{dC} \\ \dfrac{d\zeta^*}{dC} \end{bmatrix} = (A_1 + A_2 B_1 (D - C_1 A_4 B_1)^{-1} C_1 A_3) \begin{bmatrix} \dfrac{\partial \phi}{\partial C} \\ \dfrac{\partial \psi}{\partial C} \end{bmatrix}.$$

Therefore, the bottleneck of computation is the inverting step in computing K. Note L is a diagonal matrix, we can further lower the computation cost by applying the rules for inverting a block matrix again. The value of λ_3 and λ_4 can be estimated from the fact $p = 0, q = 0$. We detail the algorithm in Algorithm Algorithm 10.

Algorithm 10 Computing the gradient for w

Require: $C \in \mathbb{R}^{m\times n}, \mu, \nu, \epsilon, \frac{dC}{dw}$

Run forward pass to get $S = S^*_{\mathrm{r}}, \bar{\mu}, \bar{\nu}, \xi, \zeta$

$x_1 = \sum_{i=1}^{\lceil n/2 \rceil}(\bar{\mu}_i - \mu_i), x_2 = \sum_{i=\lceil n/2 \rceil}^{n}(\bar{\mu}_i - \mu_i), b_1 = -\sum_{i=1}^{\lceil n/2 \rceil}\xi_i, b_2 = -\sum_{i=\lceil n/2 \rceil}^{n}\xi_i$

$[\lambda_1, \lambda_3]^\top = [\lceil n/2 \rceil, x_1; n - \lceil n/2 \rceil, x_2]^{-1}[b1, b2]^\top$

$x_1 = \sum_{j=1}^{\lceil m/2 \rceil}(\bar{\nu}_j - \nu_j), x_2 = \sum_{j=\lceil m/2 \rceil}^{m}(\bar{\nu}_j - \nu_j), b_1 = -\sum_{j=1}^{\lceil m/2 \rceil}\zeta_j, b_2 = -\sum_{j=\lceil m/2 \rceil}^{m}\zeta_j$

$[\lambda_2, \lambda_4]^\top = [\lceil m/2 \rceil, x_1; m - \lceil m/2 \rceil, x_2]^{-1}[b1, b2]^\top$

$\bar{\mu} = \bar{\mu} + \epsilon(2\lambda_3 \mathbf{1}_n + \frac{\epsilon_1}{\bar{\mu}})^{-1}, \bar{\nu} = \bar{\nu} + \epsilon(2\lambda_4 \mathbf{1}_m + \frac{\epsilon_2}{\bar{\nu}})^{-1}$

$\bar{\nu}' = \bar{\nu}[:-1], S' = S[:, :-1]$

$\mathcal{K} \leftarrow \mathrm{diag}(\bar{\nu}') - (S')^T(\mathrm{diag}(\bar{\mu}))^{-1}S'$

$H_1 \leftarrow (\mathrm{diag}(\bar{\mu}))^{-1} + (\mathrm{diag}(\bar{\mu}))^{-1}S'\mathcal{K}^{-1}(S')^\top(\mathrm{diag}(\bar{\mu}))^{-1}$

$H_2 \leftarrow -(\mathrm{diag}(\bar{\mu}))^{-1}S'\mathcal{K}^{-1}$

$H_3 \leftarrow (H_2)^\top$

$H_4 \leftarrow \mathcal{K}^{-1}$

Pad H_2 to be $[n, m]$ with value 0

Pad H_3 to be $[m, n]$ with value 0

Pad H_4 to be $[m, m]$ with value 0

$L = \mathrm{diag}([\epsilon(2\lambda_3 \mathbf{1}_n + \frac{\epsilon_1}{\bar{\mu}})^{-1}, \epsilon(2\lambda_4 \mathbf{1}_m + \frac{\epsilon_2}{\bar{\nu}})^{-1}])$

$A1 = [H_1, H_2; H_3, H_4]$

$A_2 = -A_1 \cdot L$

$A_3 = A_2^\top$

$A_4 = L + L \cdot A_1 \cdot L$

$E = A_1 + A_2 \cdot B1(D - C \cdot A_4 \cdot B)^{-1}C \cdot A_3$, where $B1, C_1, D$ defined above

$[J_1, J_2; J_3, J_4] = E$, where $J_1 \in \mathbb{R}^{n\times n}, J_2 \in \mathbb{R}^{n\times m}, J_3 \in \mathbb{R}^{m\times n}, J_4 \in \mathbb{R}^{m\times m}$

$[\frac{d\xi^*}{dC}]_{nij} \leftarrow [J_1]_{ni}S_{ij} + [J_2]_{nj}S_{ij}$

$[\frac{d\zeta^*}{dC}]_{mij} \leftarrow [J_3]_{mi}S_{ij} + [J_4]_{mj}S_{ij}$

Pad $\frac{d\zeta^*}{dC}$ to be $[m, n, m]$ with value 0

$[\frac{d\mathcal{L}}{dC}]_{ij} \leftarrow \frac{1}{\epsilon}(-C_{ij}S_{ij} + \sum_{n,m} C_{nm}S_{nm}[\frac{da^*}{dC}]_{nij} + \sum_{n,m} C_{nm}S_{nm}[\frac{db^*}{dC}]_{mij}) + S_{ij}$

return $\dfrac{d\mathcal{L}}{dC}\dfrac{dC}{dw}$

D.6 Details on Experiments

D.6.1 Unlabeled Sensing

We now provide more training details for experiments in Section subsection 5.4.1. Here, AM and ROBOT is trained with batch size 500 and learning rate 10^{-4} for $2,000$ iterations. For the Sinkhorn algorithm in ROBOT we set $\epsilon = 10^{-4}$. We run RS for 2×10^5 iterations with inlier threshold as 10^{-2}. Other settings for the hyper-parameters in the baselines follows the default settings of their corresponding papers.

D.6.2 Nonlinear Regression

For the nonlinear regression experiment in Section subsection 5.4.2, ROBOT and ROBOT-robust is trained with learning rate 10^{-4} for 80 iterations. For $n = 100, 200, 500, 1000, 2000$, we set batch size $10, 30, 50, 100, 300$, respectively.We set $\epsilon = 10^{-4}$ for the Sinkhorn algorithm in ROBOT. For Oracle and LS, we perform ordinary regression model and ensure convergence, i.e., learning rate 5×10^{-2} for 100 iterations.

D.6.3 Flow Cytometry

We provide more details for the Flow Cytometry experiment in Section subsection 5.4.3. In the FC seting, ROBOT is trained with batch size 1260 and learning rate 10^{-4} for 80 iterations. In the GFC seting, ROBOT is trained with batch size 1260 and learning rate 6×10^{-4} for 60 iterations. We set $\epsilon = 10^{-4}$ for the Sinkhorn algorithm in ROBOT. Other settings for the hyper-parameters in the baselines follows the default settings of their corresponding papers. EM is initialized by AM.

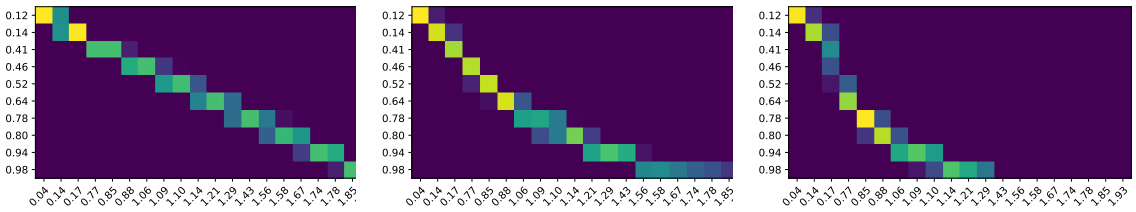
D.6.4 Multi-Object Tracking

For the MOT experiments in Section subsection 5.4.4, the reported results of `MOT17` (train) and `MOT17` (dev) is trained on `MOT17` (train), and the reported results of `MOT20` (train)

and MOT20 (dev) is trained on MOT20 (train). Each model is trained for 1 epoch. We adopt Adam optimizer with learning rate$= 10^{-5}$, $\epsilon = 10^{-4}$, and $\eta = 10^{-3}$. To track the birth and death of the tracks, we adapt the inference code of [144].

D.6.5 The Effect of ρ_1 and ρ_2

We visualize S_{r}^* computed from the robust optimal transport problem in Figure Figure D.1. The two input distributions are Unif$(0, 2)$ and Unif$(0, 1)$. We can see that with large enough ρ_1 and ρ_2, Unif$(0, 1)$ would be aligned with the first half of Unif$(0, 2)$.



(a) $\rho_1 = 0, \rho_2 = 0$ (b) $\rho_1 = 0.1, \rho_2 = 0.1$ (c) $\rho_1 = 0.2, \rho_2 = 0.2$

Figure D.1: Computed S^* for robust optimal transport problem.

D.6.6 Comparison of Residuals in Linear Regression

Settings. We generate n data points $\{(y_i, [x_i, z_i])\}_{i=1}^n$, where $x_i \in \mathbb{R}^d$ and $z_i \in \mathbb{R}^e$. We first generate $x_i \sim \mathcal{N}(\mathbf{0}_d, \boldsymbol{I}_d)$, $z_i \sim \mathcal{N}(\mathbf{0}_e, \boldsymbol{I}_e)$, $w \sim \mathcal{N}(\mathbf{0}_{d+e}, \boldsymbol{I}_{d+e})$, and $\varepsilon_i \sim \mathcal{N}(0, \rho_{\text{noise}}^2)$. Then we compute $y_i = f([x_i, z_i]; w) + \varepsilon_i$. Next, we randomly permute the order of $\{z_i\}$ so that we lose the data correspondence. Here, $\mathcal{D}_1 = \{(x_i, y_i)\}$ and $\mathcal{D}_2 = \{z_j\}$ mimic two parts of data collected from two separate platforms.

We adopt a linear model $f(x; w) = x^\top w$. To evaluate model performance, we use error$= \sum_i (\widehat{y}_i - y_i)^2 / \sum_i (y_i - \bar{y})^2$, where $\widehat{y}_i$ is the predicted label, and $\bar{y}$ is the mean of $\{y_i\}$.

Baselines. We use Oracle, LS, Stochastic-EM as the baselines. Notice that without a proper

initialization, Stochastic-EM performs well in partially permuted cases, but not in fully shuffled cases. For better visualization, we only include this baseline in one experiment. Furthermore, we adopt two new baselines: Sliced-GW [168] and Sinkhorn-GW [169], which can be used to align distributions and points sets.

Results. We visualize the fitting error of regression models in Figure Figure D.2. We can see that ROBOT outperforms all the baselines except Oracle. Also, our model can beat the Oracle model when the dimension is low or when the noise is large.

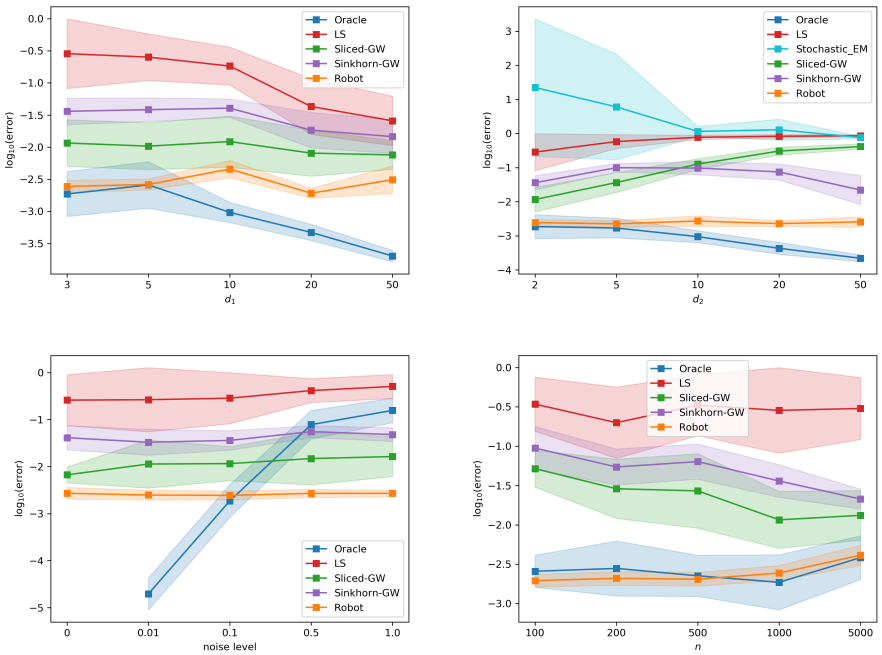


Figure D.2: Linear regression. We use $n = 1000$, $d = 2$, $e = 3$, $\rho^2_{\text{noise}} = 0.1$ as defaults.

REFERENCES

[1] G. Monge, "Mémoire sur la théorie des déblais et des remblais," *Histoire de l'Académie Royale des Sciences de Paris*, 1781.

[2] M. Cuturi, "Sinkhorn distances: Lightspeed computation of optimal transport," in *Advances in neural information processing systems*, 2013, pp. 2292–2300.

[3] J.-D. Benamou, G. Carlier, M. Cuturi, L. Nenna, and G. Peyré, "Iterative bregman projections for regularized transportation problems," *SIAM Journal on Scientific Computing*, vol. 37, no. 2, A1111–A1138, 2015.

[4] J. Altschuler, J. Weed, and P. Rigollet, "Near-linear time approximation algorithms for optimal transport via Sinkhorn iteration," pp. 1964–1974, 2017.

[5] L. Chizat, G. Peyré, B. Schmitzer, and F.-X. Vialard, "Scaling algorithms for unbalanced transport problems," *arXiv preprint arXiv:1607.05816*, 2016.

[6] M. Mandad, D. Cohen-Steiner, L. Kobbelt, P. Alliez, and M. Desbrun, "Variance-minimizing transport plans for inter-surface mapping," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 39, 2017.

[7] J. Franklin and J. Lorenz, "On the scaling of multidimensional matrices," *Linear Algebra and its Applications*, vol. 114, pp. 717–735, 1989.

[8] A. Genevay, G. Peyré, and M. Cuturi, "Sinkhorn-AutoDiff: Tractable Wasserstein learning of generative models," *arXiv preprint arXiv:1706.00292*, 2017.

[9] M. Martinez, M. Haurilet, Z. Al-Halah, M. Tapaswi, and R. Stiefelhagen, "Relaxed earth mover's distances for chain-and tree-connected spaces and their use as a loss function in deep learning," *arXiv preprint arXiv:1611.07573*, 2016.

[10] J. Solomon, R. Rustamov, L. Guibas, and A. Butscher, "Wasserstein propagation for semi-supervised learning," in *International Conference on Machine Learning*, 2014, pp. 306–314.

[11] L. Kantorovich, "On mass transfer problem," *Doklady Akademii Nauk SSSR37*, pp. 199–201, 1942.

[12] J. Rabin, S. Ferradans, and N. Papadakis, "Adaptive color transfer with relaxed optimal transport," in *IEEE International Conference on Image Processing*, IEEE, 2014, pp. 4852–4856.

[13] O. Pele and M. Werman, "Fast and robust earth mover's distances," in *IEEE 12th International Conference on Computer Vision*, IEEE, 2009, pp. 460–467.

[14] P. Dvurechensky, S. Omelchenko, and A. Tiurin, "Adaptive similar triangles method: A stable alternative to Sinkhorn's algorithm for regularized optimal transport," *arXiv preprint arXiv:1706.07622*, 2017.

[15] A. Thibault, L. Chizat, C. Dossal, and N. Papadakis, "Overrelaxed Sinkhorn-Knopp algorithm for regularized optimal transport," *arXiv preprint arXiv:1711.01851*, 2017.

[16] S. Afriat, "Theory of maxima and the method of Lagrange," *SIAM Journal on Applied Mathematics*, vol. 20, no. 3, pp. 343–357, 1971.

[17] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends in Optimization*, vol. 1, no. 3, pp. 127–239, 2014.

[18] R. Rockafellar, "Augmented Lagrangians and applications of the proximal point algorithm in convex programming," *Mathematics of Operations Research*, vol. 1, no. 2, pp. 97–116, 1976.

[19] ——, "Monotone operators and the proximal point algorithm," *SIAM Journal on Control and Optimization*, vol. 14, no. 5, pp. 877–898, 1976.

[20] M. Solodov and B. Svaiter, "A unified framework for some inexact proximal point algorithms," *Numerical Functional Analysis and Optimization*, vol. 22, no. 7-8, pp. 1013–1035, 2001.

[21] M. Schmidt, N. Roux, and F. Bach, "Convergence rates of inexact proximal-gradient methods for convex optimization," in *Advances in Neural Information Processing Systems*, 2011, pp. 1458–1466.

[22] R. Flamary and N. Courty, *Pot python optimal transport library*, 2017.

[23] B. Schmitzer, "Stabilized sparse scaling algorithms for entropy regularized transport problems," *arXiv preprint arXiv:1610.06519*, 2016.

[24] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[25] E. Reinhard, M. Adhikhmin, B. Gooch, and P. Shirley, "Color transfer between images," *IEEE Computer Graphics and Applications*, vol. 21, no. 5, pp. 34–41, 2001.

[26] X. Xiao and L. Ma, "Color transfer in correlated color space," in *ACM international Conference on Virtual Reality Continuum and its Applications*, ACM, 2006, pp. 305–309.

[27] Y. Rubner, C. Tomasi, and L. J. Guibas, "A metric for distributions with applications to image databases," in *Computer Vision, 1998. Sixth International Conference on*, IEEE, 1998, pp. 59–66.

[28] J. Rabin, G. Peyré, J. Delon, and M. Bernot, "Wasserstein barycenter and its application to texture mixing," in *International Conference on Scale Space and Variational Methods in Computer Vision*, Springer, 2011, pp. 435–446.

[29] M. Cuturi and A. Doucet, "Fast computation of Wasserstein barycenters," in *International Conference on Machine Learning*, 2014, pp. 685–693.

[30] J. Solomon, F. De Goes, G. Peyré, M. Cuturi, A. Butscher, A. Nguyen, T. Du, and L. Guibas, "Convolutional wasserstein distances: Efficient optimal transportation on geometric domains," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, p. 66, 2015.

[31] M. Staib, S. Claici, J. Solomon, and S. Jegelka, "Parallel streaming Wasserstein barycenters," *arXiv preprint arXiv:1705.07443*, 2017.

[32] S. Claici, E. Chien, and J. Solomon, "Stochastic wasserstein barycenters," *arXiv preprint arXiv:1802.05757*, 2018.

[33] A. Genevay, M. Cuturi, G. Peyré, and F. Bach, "Stochastic optimization for large-scale optimal transport," in *Advances in Neural Information Processing Systems*, 2016, pp. 3440–3448.

[34] V. Seguy, B. B. Damodaran, R. Flamary, N. Courty, A. Rolet, and M. Blondel, "Large-scale optimal transport and mapping estimation," *arXiv preprint arXiv:1711.02283*, 2017.

[35] K. D. Yang and C. Uhler, "Scalable unbalanced optimal transport using generative adversarial networks," *arXiv preprint arXiv:1810.11447*, 2018.

[36] A. Brock, J. Donahue, and K. Simonyan, "Large scale gan training for high fidelity natural image synthesis," *arXiv preprint arXiv:1809.11096*, 2018.

[37] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," *arXiv preprint arXiv:1611.03530*, 2016.

[38] C. Li, H. Farkhoor, R. Liu, and J. Yosinski, "Measuring the intrinsic dimension of objective landscapes," *arXiv preprint arXiv:1804.08838*, 2018.

[39] M.-Y. Liu and O. Tuzel, "Coupled generative adversarial networks," in *Advances in neural information processing systems*, 2016, pp. 469–477.

[40] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," *arXiv preprint arXiv:1806.07366*, 2018.

[41] L. Chizat, G. Peyré, B. Schmitzer, and F.-X. Vialard, "Unbalanced optimal transport: Geometry and kantorovich formulation," *arXiv preprint arXiv:1508.05216*, 2015.

[42] C. Frogner, C. Zhang, H. Mobahi, M. Araya, and T. A. Poggio, "Learning with a wasserstein loss," in *Advances in Neural Information Processing Systems*, 2015, pp. 2053–2061.

[43] Y. Xie, X. Wang, R. Wang, and H. Zha, "A fast proximal point method for wasserstein distance," *arXiv preprint arXiv:1802.04307*, 2018.

[44] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[45] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[46] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[47] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," in *Advances in neural information processing systems*, 2016, pp. 2172–2180.

[48] J. Zhao, M. Mathieu, and Y. LeCun, "Energy-based generative adversarial network," *arXiv preprint arXiv:1609.03126*, 2016.

[49] Z. Dai, A. Almahairi, P. Bachman, E. Hovy, and A. Courville, "Calibrating energy-based generative adversarial networks," *arXiv preprint arXiv:1702.01691*, 2017.

[50] H. Jiang, Z. Chen, M. Chen, F. Liu, D. Wang, and T. Zhao, "On computation and generalization of gans with spectrum control," *arXiv preprint arXiv:1812.10912*, 2018.

[51] L. Dinh, D. Krueger, and Y. Bengio, "Nice: Non-linear independent components estimation," *arXiv preprint arXiv:1410.8516*, 2014.

[52] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," *arXiv preprint arXiv:1605.08803*, 2016.

[53] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," in *Advances in Neural Information Processing Systems*, 2018, pp. 10 235–10 244.

[54] W. Grathwohl, R. T. Chen, J. Betterncourt, I. Sutskever, and D. Duvenaud, "Ffjord: Free-form continuous dynamics for scalable reversible generative models," *arXiv preprint arXiv:1810.01367*, 2018.

[55] C. Villani, *Optimal transport: old and new*. Springer Science & Business Media, 2008, vol. 338.

[56] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," *arXiv preprint arXiv:1802.05957*, 2018.

[57] G. H. Golub and H. A. Van der Vorst, "Eigenvalue computation in the 20th century," in *Numerical analysis: historical developments in the 20th century*, Elsevier, 2001, pp. 209–239.

[58] Y. Liu, Z. Wang, H. Jin, and I. Wassell, "Multi-task adversarial network for disentangled feature learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3743–3751.

[59] B. Pass, "Multi-marginal optimal transport: Theory and applications," *ESAIM: Mathematical Modelling and Numerical Analysis*, vol. 49, no. 6, pp. 1771–1790, 2015.

[60] S. Erlander and N. F. Stewart, *The gravity model in transportation analysis: theory and extensions*. Vsp, 1990, vol. 3.

[61] P. J. Davis and P. Rabinowitz, *Methods of numerical integration*. Courier Corporation, 2007.

[62] N. Courty, R. Flamary, A. Habrard, and A. Rakotomamonjy, "Joint distribution optimal transportation for domain adaptation," in *Advances in Neural Information Processing Systems*, 2017, pp. 3730–3739.

[63] B. B. Damodaran, B. Kellenberger, R. Flamary, D. Tuia, and N. Courty, "Deepjdot: Deep joint distribution optimal transport for unsupervised domain adaptation," *arXiv preprint arXiv:1803.10081*, 2018.

[64] V. Seguy, B. B. Damodaran, R. Flamary, N. Courty, A. Rolet, and M. Blondel, "Large scale optimal transport and mapping estimation," in *International Conference on Learning Representations*, 2018.

[65] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[66] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, 2013, p. 3.

[67] S. T. Rachev, "The monge–kantorovich mass transference problem and its stochastic applications," *Theory of Probability & Its Applications*, vol. 29, no. 4, pp. 647–676, 1985.

[68] L. Onural, "Impulse functions over curves and surfaces and their applications to diffraction," *Journal of mathematical analysis and applications*, vol. 322, no. 1, pp. 18–27, 2006.

[69] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[70] Y. Ganin and V. Lempitsky, "Unsupervised domain adaptation by backpropagation," *arXiv preprint arXiv:1409.7495*, 2014.

[71] G. Peyré, M. Cuturi, *et al.*, "Computational optimal transport," Tech. Rep., 2017.

[72] P. Li, X. Liang, D. Jia, and E. P. Xing, "Semantic-aware grad-gan for virtual-to-real urban scene adaption," *arXiv preprint arXiv:1801.01726*, 2018.

[73] I. Sobel, "An isotropic 3$\times$ 3 image gradient operater," *Machine vision for three-dimensional scenes*, pp. 376–379, 1990.

[74] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," *arXiv preprint*, 2017.

[75] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *arXiv preprint*, 2017.

[76] J. J. Hull, "A database for handwritten text recognition research," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 16, no. 5, pp. 550–554, 1994.

[77] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS workshop on deep learning and unsupervised feature learning*, vol. 2011, 2011, p. 5.

[78] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[79] A. Babenko, A. Slesarev, A. Chigorin, and V. Lempitsky, "Neural codes for image retrieval," in *European conference on computer vision*, Springer, 2014, pp. 584–599.

[80] F. Radenović, G. Tolias, and O. Chum, "Cnn image retrieval learns from bow: Unsupervised fine-tuning with hard examples," in *European conference on computer vision*, Springer, 2016, pp. 3–20.

[81] A. Gordo, J. Almazán, J. Revaud, and D. Larlus, "Deep image retrieval: Learning global representations for image search," in *European conference on computer vision*, Springer, 2016, pp. 241–257.

[82] D. R. Reddy *et al.*, *Speech understanding systems: A summary of results of the five-year research effort. department of computer science*, 1977.

[83] S. Wiseman and A. M. Rush, "Sequence-to-sequence learning as beam-search optimization," *arXiv preprint arXiv:1606.02960*, 2016.

[84] C. A. Hoare, "Algorithm 65: Find.," *Commun. ACM*, vol. 4, no. 7, pp. 321–322, Jul. 1961.

[85] N. Papernot and P. McDaniel, "Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning," *arXiv preprint arXiv:1803.04765*, 2018.

[86] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra, "Efficient projections onto the l 1-ball for learning in high dimensions," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 272–279.

[87] A. Griewank and A. Walther, *Evaluating derivatives: principles and techniques of algorithmic differentiation*. Siam, 2008, vol. 105.

[88] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, pp. 136–145.

[89] G. Luise, A. Rudi, M. Pontil, and C. Ciliberto, "Differential properties of sinkhorn approximation for learning with wasserstein distance," in *Advances in Neural Information Processing Systems*, 2018, pp. 5859–5870.

[90] L. V. Kantorovich, "Mathematical methods of organizing and planning production," *Management science*, vol. 6, no. 4, pp. 366–422, 1960.

[91] R. Sinkhorn and P. Knopp, "Concerning nonnegative matrices and doubly stochastic matrices," *Pacific Journal of Mathematics*, vol. 21, no. 2, pp. 343–348, 1967.

[92] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[93] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[94] M. Cuturi, O. Teboul, and J.-P. Vert, "Differentiable ranking and sorting using optimal transport," in *Advances in Neural Information Processing Systems*, 2019, pp. 6858–6868.

[95] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[96] A. Grover, E. Wang, A. Zweig, and S. Ermon, "Stochastic optimization of sorting networks via continuous relaxations," *arXiv preprint arXiv:1903.08850*, 2019.

[97] S. M. Xie and S. Ermon, "Reparameterizable subset sampling via continuous relaxations," in *International Joint Conference on Artificial Intelligence*, 2019.

[98] T. Plötz and S. Roth, "Neural nearest neighbors networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 1087–1098.

[99] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 1171–1179.

[100] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[101] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," *arXiv preprint arXiv:1508.07909*, 2015.

[102] M.-T. Luong, I. Sutskever, Q. V. Le, O. Vinyals, and W. Zaremba, "Addressing the rare word problem in neural machine translation," *arXiv preprint arXiv:1410.8206*, 2014.

[103] N. Durrani, B. Haddow, P. Koehn, and K. Heafield, "Edinburgh's phrase-based machine translation systems for wmt-14," in *Proceedings of the Ninth Workshop on Statistical Machine Translation*, 2014, pp. 97–104.

[104] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[105] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[106] S. Jean, K. Cho, R. Memisevic, and Y. Bengio, "On using very large target vocabulary for neural machine translation," *arXiv preprint arXiv:1412.2007*, 2014.

[107] C. Zhu, X. Tan, F. Zhou, X. Liu, K. Yue, E. Ding, and Y. Ma, "Fine-grained video categorization with redundancy reduction attention," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 136–152.

[108] J. Schlemper, O. Oktay, M. Schaap, M. Heinrich, B. Kainz, B. Glocker, and D. Rueckert, "Attention gated networks: Learning to leverage salient regions in medical images," *Medical image analysis*, vol. 53, pp. 197–207, 2019.

[109] S. Shankar, S. Garg, and S. Sarawagi, "Surprisingly easy hard-attention for sequence to sequence learning," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 640–645.

[110] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush, "OpenNMT: Open-source toolkit for neural machine translation," in *Proc. ACL*, 2017.

[111] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[112] W. Kool, H. Van Hoof, and M. Welling, "Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement," *arXiv preprint arXiv:1903.06059*, 2019.

[113] J. M. Stanton, "Galton, pearson, and the peas: A brief history of linear regression for statistics instructors," *Journal of Statistics Education*, vol. 9, no. 3, 2001.

[114] A. Abid and J. Zou, "A stochastic expectation-maximization approach to shuffled linear regression," in *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, IEEE, 2018, pp. 470–477.

[115] J. Unnikrishnan, S. Haghighatshoar, and M. Vetterli, "Unlabeled sensing with random linear measurements," *IEEE Transactions on Information Theory*, vol. 64, no. 5, pp. 3237–3253, 2018.

[116] A. Pananjady, M. J. Wainwright, and T. A. Courtade, "Linear regression with an unknown permutation: Statistical and computational limits," in *2016 54th Annual Allerton Conference on Communication, Control, and Computing*, 2016, pp. 417–424.

[117] ——, "Linear regression with shuffled data: Statistical and computational limits of permutation recovery," *IEEE Transactions on Information Theory*, vol. 64, no. 5, pp. 3286–3300, 2017.

[118] A. Abid, A. Poon, and J. Zou, "Linear regression with shuffled labels," *arXiv preprint arXiv:1705.01342*, 2017.

[119] G. Elhami, A. J. Benjamin, B. Haro, and M. Vetterli, "Unlabeled sensing: Reconstruction algorithm and theoretical guarantees," in *2017 42nd IEEE International Conference on Acoustics, Speech and Signal Processing*, 2017.

[120] D. J. Hsu, K. Shi, and X. Sun, "Linear regression without correspondence," in *Advances in Neural Information Processing Systems*, 2017, pp. 1531–1540.

[121] M. Tsakiris and L. Peng, "Homomorphic sensing," in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, Long Beach, California, USA: PMLR, Sep. 2019, pp. 6335–6344.

[122] A. Pananjady, M. J. Wainwright, and T. A. Courtade, "Denoising linear models with permuted data," in *2017 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2017, pp. 446–450.

[123] M. C. Tsakiris, L. Peng, A. Conca, L. Kneip, Y. Shi, H. Choi, *et al.*, "An algebraic-geometric approach to shuffled linear regression," *arXiv preprint arXiv:1810.05440*, 2018.

[124] L. Peng and M. C. Tsakiris, "Linear regression without correspondences via concave minimization," *arXiv preprint arXiv:2003.07706*, 2020.

[125] E. Varol and A. Nejatbakhsh, "Robust approximate linear regression without correspondence," *arXiv preprint arXiv:1906.00273*, 2019.

[126] M. Liero, A. Mielke, and G. Savaré, "Optimal entropy-transport problems and a new hellinger–kantorovich distance between positive measures," *Inventiones mathematicae*, vol. 211, no. 3, pp. 969–1117, 2018.

[127] G. Birkhoff, "Three observations on linear algebra," *Univ. Nac. Tacuman, Rev. Ser. A*, vol. 5, pp. 147–151, 1946.

[128] J. Von Neumann, "A certain zero-sum two-person game equivalent to the optimal assignment problem," *Contributions to the Theory of Games*, vol. 2, no. 0, pp. 5–12, 1953.

[129] M. Marcus and R. Ree, "Diagonals of doubly stochastic matrices," *The Quarterly Journal of Mathematics*, vol. 10, no. 1, pp. 296–302, 1959.

[130] G. M. Ziegler, *Lectures on polytopes*. Springer Science & Business Media, 2012, vol. 152.

[131] G. B. Dantzig, *Linear programming and extensions*. Princeton university press, 1998, vol. 48.

[132] S. Kondratyev, L. Monsaingeon, D. Vorotnikov, *et al.*, "A new optimal transport distance on the space of finite radon measures," *Advances in Differential Equations*, vol. 21, no. 11/12, pp. 1117–1164, 2016.

[133] L. Chizat, G. Peyré, B. Schmitzer, and F.-X. Vialard, "An interpolating distance between optimal transport and fisher–rao metrics," *Foundations of Computational Mathematics*, vol. 18, no. 1, pp. 1–44, 2018.

[134] L. Chizat, G. Peyré, B. Schmitzer, and F.-X. Vialard, "Unbalanced optimal transport: Dynamic and kantorovich formulations," *Journal of Functional Analysis*, vol. 274, no. 11, pp. 3090–3123, 2018.

[135] L. Chizat, G. Peyré, B. Schmitzer, and F.-X. Vialard, "Scaling algorithms for unbalanced optimal transport problems," *Mathematics of Computation*, vol. 87, no. 314, pp. 2563–2609, 2018.

[136] M. Slawski and E. Ben-David, "Linear regression with sparsely permuted data," *Electronic Journal of Statistics*, 2019.

[137] M. Slawski, E. Ben-David, and P. Li., "A two-stage approach to multivariate linear regression with sparsely mismatched data," *arXiv preprint arXiv:1907.07148*, 2019.

[138] C. G. Knight, M. Platt, W. Rowe, D. C. Wedge, F. Khan, P. J. Day, A. McShea, J. Knowles, and D. B. Kell, "Array-based evolution of dna aptamers allows modelling of an explicit sequence-fitness landscape," *Nucleic acids research*, vol. 37, no. 1, e6–e6, 2009.

[139] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, pp. 1627–1645, 2009.

[140] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[141] Z. He, J. Li, D. Liu, H. He, and D. Barber, "Tracking by animation: Unsupervised learning of multi-object attentive trackers," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 1318–1327.

[142] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler, "MOT16: A benchmark for multi-object tracking," *arXiv:1603.00831 [cs]*, Mar. 2016, arXiv: 1603.00831.

[143] P. Dendorfer, H. Rezatofighi, A. Milan, J. Shi, D. Cremers, I. Reid, S. Roth, K. Schindler, and L. Leal-Taixé, "Mot20: A benchmark for multi object tracking in crowded scenes," *arXiv:2003.09003[cs]*, Mar. 2020, arXiv: 2003.09003.

[144] Y. Xu, Y. Ban, X. Alameda-Pineda, and R. Horaud, "Deepmot: A differentiable framework for training multiple object trackers," *arXiv preprint arXiv:1906.06618*, 2019.

[145] B. Li, J. Yan, W. Wu, Z. Zhu, and X. Hu, "High performance visual tracking with siamese region proposal network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8971–8980.

[146] E. Ristani, F. Solera, R. Zou, R. Cucchiara, and C. Tomasi, "Performance measures and a data set for multi-target, multi-camera tracking," in *European Conference on Computer Vision*, Springer, 2016, pp. 17–35.

[147] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *2016 IEEE International Conference on Image Processing (ICIP)*, IEEE, 2016, pp. 3464–3468.

[148] S. Haghighatshoar and G. Caire, "Signal recovery from unlabeled samples," *IEEE Transactions on Signal Processing*, vol. 66, no. 5, pp. 1242–1257, 2017.

[149] P. Rigollet and J. Weed, "Uncoupled isotonic regression via minimum wasserstein deconvolution," *arXiv preprint arXiv:1806.10648*, 2018.

[150] X. Shi, X. Lu, and T. Cai, "Spherical regresion under mismatch corruption with application to automated knowledge translation," *arXiv preprint arXiv:1810.05679*, 2018.

[151] M. Slawski, M. Rahmani, and P. Li, "A sparse representation-based approach to linear regression with partially shuffled labels," in *35th Conference on Uncertainty in Artificial Intelligence, UAI 2019*, 2019.

[152] S. Thrun, "Simultaneous localization and mapping," in *Robotics and cognitive approaches to spatial mapping*, Springer, 2007, pp. 13–41.

[153] W. S. Robinson, "A method for chronologically ordering archaeological deposits," *American antiquity*, vol. 16, no. 4, pp. 293–301, 1951.

[154] L. Keller, M. J. Siavoshani, C. Fragouli, K. Argyraki, and S. Diggavi, "Identity aware sensor networks," in *IEEE INFOCOM 2009*, IEEE, 2009, pp. 2177–2185.

[155] P. David, D. Dementhon, R. Duraiswami, and H. Samet, "Softposit: Simultaneous pose and correspondence determination," *International Journal of Computer Vision*, vol. 59, no. 3, pp. 259–284, 2004.

[156] X. Huang and A. Madan, "Cap3: A dna sequence assembly program," *Genome research*, vol. 9, no. 9, pp. 868–877, 1999.

[157] F. Bassetti, A. Bodini, and E. Regazzini, "On minimum kantorovich distance estimators," *Statistics & probability letters*, vol. 76, no. 12, pp. 1298–1302, 2006.

[158] J. Eckstein, "Nonlinear proximal point algorithms using bregman functions, with applications to convex programming," *Mathematics of Operations Research*, vol. 18, no. 1, pp. 202–226, 1993.

[159] ——, "Approximate iterations in bregman-function-based proximal algorithms," *Mathematical programming*, vol. 83, no. 1-3, pp. 113–123, 1998.

[160] M. Teboulle, "Entropic proximal mappings with applications to nonlinear programming," *Mathematics of Operations Research*, vol. 17, no. 3, pp. 670–690, 1992.

[161] S. M. Robinson, "Some continuity properties of polyhedral multifunctions," in *Mathematical Programming at Oberwolfach*, Springer, 1981, pp. 206–214.

[162] B. Polyak, *Introduction to optimization*. Optimization Software Inc., New York, 1987.

[163] A. M.-C. So and Z. Zhou, "Non-asymptotic convergence analysis of inexact gradient methods for machine learning without strong convexity," *Optimization Methods and Software*, vol. 32, no. 4, pp. 963–992, 2017.

[164] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[165] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *Siam Review*, vol. 60, no. 2, pp. 223–311, 2018.

[166] Y. Xie, H. Dai, M. Chen, B. Dai, T. Zhao, H. Zha, W. Wei, and T. Pfister, "Differentiable top-k operator with optimal transport," *arXiv preprint arXiv:2002.06504*, 2020.

[167] M. Wang, Y. Chen, J. Liu, and Y. Gu, "Random multi-constraint projection: Stochastic gradient methods for convex optimization with many constraints," *arXiv preprint arXiv:1511.03760*, 2015.

[168] T. Vayer, R. Flamary, R. Tavenard, L. Chapel, and N. Courty, "Sliced gromov-wasserstein," *arXiv preprint arXiv:1905.10124*, 2019.

[169] H. Xu, D. Luo, H. Zha, and L. Carin, "Gromov-wasserstein learning for graph matching and node embedding," *arXiv preprint arXiv:1901.06003*, 2019.